Computational Linear Algebra

ANSWERS FOR CODING ASSIGNMENT

Submitted By
Rahul Madhavan
Student, MTech AI

Helpers

Throughout all the files, we make use of certain helper functions. At the outset, we detail them for ease of programming notation later

1. Utilities.py

```python
import numpy as np

def L1Norm(vector):
    return np.sum(np.abs(vector))

def LInfNorm(vector):
    return np.max(np.abs(vector))

def sumSquares(vector):
    return np.sum(np.square(vector))

def L2Norm(vector):
    return np.sqrt(sumSquares(vector))
```

2. randomMatrix.py

```python
import numpy as np
import numpy.random as random


def generateUniformRandomMatrices(rows, columns, lo=-1, hi=1):
    return random.uniform(lo, hi, (rows, columns))


def generateStdNormalRandomMatrices(rows, columns):
    return random.randn(rows, columns)


def sparseRandomNormalMatrix(rows, columns, density):
    A = generateStdNormalRandomMatrices(rows, columns)
    sparseA = A.copy()
    for i in range(rows):
        for j in range(columns):
            r = random.uniform(0, 1)
            if (r > density):
                sparseA[i, j] = 0
    return sparseA


def gendata_lasso(m=500, n=2500, noise=0, option=1):
    # function to generate test data for lasso
    #   Input:  m: no. of observations
    #           n: no. of features
    #       noise: standard deviation
    #      option: 0: no noise
    #              1: noise added by gaussian distribution
    #              2: noise added as an outlier (selecting any 1 of the
    #                 observations)
    ##
    x0 = sparseRandomNormalMatrix(n, 1, 0.05)
    A = generateStdNormalRandomMatrices(m, n)
    # normalize columns
    ANormalizer = np.square(A)
    ANormalizer = np.sum(ANormalizer, axis=0)
    ANormalizer = np.sqrt(ANormalizer)
```

```
    ANormalizer = 1 / ANormalizer
    A = A.dot(np.diag(ANormalizer))

    v = np.sqrt(0.001) * generateStdNormalRandomMatrices(m, 1)
    b = A.dot(x0) + v

    if option == 1:
        b = b + noise * random.rand(b.shape[0], b.shape[1])
        return A, b

    if option == 2:
        randomRow = random.randint(m)
        b[randomRow] = b[randomRow] + noise * random.uniform(0, 1)
        return A, b

    return A, b


def generateLowRank(m, n, rank):
    randomMatrix = np.random.rand(m, n)
    U, Diag, V = np.linalg.svd(randomMatrix)
    Diag[rank:] = Diag[rank:] * 0
    out = (U @ np.diag(Diag)) @ V
    return out


def rgg(num_vertices=5, lo=1, hi=10, density=0.5):
    if num_vertices <= 0:
        print("No. of vertices must be positive")
        return

    Weight = lo + (hi - lo + 1) * generateUniformRandomMatrices(num_vertices,
num_vertices, lo=0, hi=1)
    Weight = 0.5 * (Weight + Weight.T)

    probMat = generateUniformRandomMatrices(num_vertices, num_vertices, lo=0, hi=1)
    Connectivity = probMat >= density
    Connectivity = np.triu(Connectivity, 1)
    Connectivity = Connectivity + Connectivity.T
    adjacencyMatrix = np.multiply(Connectivity, Weight)
    return adjacencyMatrix


def generateCompleteBipartite(k1, k2):
    numNodes = k1 + k2
    list_of_nodes = list(range(numNodes))
    np.random.shuffle(list_of_nodes)
    first_part = list_of_nodes[:k1]
    second_part = list_of_nodes[k1:]
    out = np.zeros((numNodes, numNodes))
    for f in first_part:
        for s in second_part:
            out[f, s] = 1
            out[s, f] = 1
    return out


def svm_gendata(Np, Nn,distance):
    Xp = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ random.randn(2, Np)
    Xp[0, :] = Xp[0, :] + distance
    yp = np.ones(Np)

    Xn = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ random.randn(2, Nn)
    Xn[0, :] = Xn[0, :] - distance
```

```python
    yn = - np.ones(Nn)

    X = np.hstack((Xp, Xn))
    y = np.hstack((yp, yn))

    return X, y

def svm_gendata2(Np, Nn,distance):
    Xp = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ random.randn(2, Np)
    Xp[0, :] = Xp[0, :] + distance
    Xp[1, :] = Xp[1, :] - distance
    yp = np.ones(Np)

    Xn = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ random.randn(2, Nn)
    Xn[0, :] = Xn[0, :] - distance
    Xn[1, :] = Xn[1, :] + distance

    yn = - np.ones(Nn)

    X = np.hstack((Xp, Xn))
    y = np.hstack((yp, yn))

    return X, y
```

Remarks

1. Notice that we have recreated ALL the Matlab functions again in Python for convenience of use. This enabled the user to fully tweak the functions as per requirement
2. We have also created wrappers for several numpy functions so that we can change the default values of the original functions as per our requirements
3. Many of the functions have been optimized for speed of use. Generally, one may think of a call to a vector generator as being O(n) and matrix generator as O(n^2), but the constants are very low as memory access is in constant time of nanoseconds.

1. Solving Norm as LPs

Method

Since this was the first Question, I tried using CVXOPT. While many of the features are quite good in CVXOPT, it is overall not as suitable for the assignment as CVXPY. So this is the only part of the assignment that has been done using CVXOPT, and the rest of the assignment often uses CVXPY with CVXOPT or MOSEK used as the solver

Implementation Notes

PART 1

```
# https://math.stackexchange.com/questions/1639716/how-can-l-1-norm-minimization-with-linear-
equality-constraints-basis-pu
We formulate the LP as minimize
sum(t_i) for |A_ix - b_i| \leq t_i

or in other words
minimize sum(t_i) for (A_ix - b_i) \leq t_i and (A_ix - b_i) \geq -t_i


Which can be written as
minimize 1.T.dot(t) for (A_ix - b_i) \leq t_i and (A_ix - b_i) \geq -t_i


OR minimize 0.T.dot(x) + 1.T.dot(t)
such that
(A_i x - t_i) \leq +b_i
\forall i in [n]
and
-(A_ix + t_i) \leq -b_i
and
-t_i < 0
\forall i in [n], where x \in R^r, t in R^n


#
OR

minimize 0.T.dot(x) + 1.T.dot(t)
such that
(A x - I t) \leq b
and
-Ax - I t \leq -b
and
0x - I t \leq 0
```

PART II

```
In the above formulation, instead of a vector t, we can just use a scalar t
Then we write

minimize t
such that
(A_ix - b_i) \leq t
and
(A_ix - b_i) \geq -t
and
t \geq 0



The problem can be restated as

minimize t
such that
(A_ix - t) \leq b_i
and
(-A_ix - t) \leq -b_i


OR

minimize 0.dot(x) + t
such that
(Ax - t1) \leq b
and
(-Ax - t1) \leq -b
```

Outputs

```
     pcost       dcost       gap    pres   dres   k/t
 0:  0.0000e+00  1.1102e-16  4e+02  4e+00  1e-16  1e+00
 1:  4.4132e+01  4.4134e+01  7e+01  6e-01  6e-16  2e-01
 2:  7.8640e+01  7.8641e+01  2e+01  2e-01  4e-15  4e-02
 3:  8.5193e+01  8.5193e+01  6e+00  5e-02  5e-15  1e-02
 4:  8.7495e+01  8.7496e+01  2e+00  2e-02  5e-15  5e-03
 5:  8.8207e+01  8.8207e+01  7e-01  6e-03  9e-15  2e-03
 6:  8.8471e+01  8.8471e+01  1e-01  1e-03  4e-15  3e-04
 7:  8.8515e+01  8.8515e+01  3e-02  3e-04  4e-15  9e-05
 8:  8.8527e+01  8.8527e+01  1e-02  9e-05  3e-14  3e-05
 9:  8.8531e+01  8.8531e+01  5e-04  4e-06  3e-14  1e-06
10:  8.8531e+01  8.8531e+01  5e-06  4e-08  3e-14  1e-08
Optimal solution found.

----------------------------------------------------------------------------------------------
--------

Results for l1Norm though LP

----------------------------------------------------------------------------------------------
--------

x= [-0.15145901 -0.04173573 -0.04512145  0.05624462 -0.07281424  0.26913976
  0.08056713 -0.03613971  0.00596245  0.15144689]

---------------------------
```

```
A.dot(x) - b = [-1.21967741 -0.25450525 -0.12747449 ...  0.55274729  0.43633053
  0.21275996]
L1Norm(A.dot(x)-b) =  88.5313506065077
LInfNorm(A.dot(x)-b) =  1.3605621194599204
L2Norm(A.dot(x)-b) =  7.746547475037685

----------------------------------------------------------------------------------------------------
--------


----------------------------------------------------------------------------------------------------
--------

Results for l1Norm though CVXPY

----------------------------------------------------------------------------------------------------
--------

x= [-0.15145901 -0.04173573 -0.04512145  0.05624462 -0.07281424  0.26913976
  0.08056713 -0.03613971  0.00596245  0.15144689]

---------------------------

A.dot(x) - b = [-1.21967741 -0.25450525 -0.12747449 ...  0.55274729  0.43633053
  0.21275996]
L1Norm(A.dot(x)-b) =  88.53135060650769
LInfNorm(A.dot(x)-b) =  1.36056211945992
L2Norm(A.dot(x)-b) =  7.746547475037685

----------------------------------------------------------------------------------------------------
--------

     pcost       dcost       gap    pres   dres   k/t
 0: -3.2027e-18  4.7705e-18  2e+00  4e+00  4e-16  1e+00
 1:  2.9759e-01  2.1593e-01  9e-01  1e+00  2e-16  3e-01
 2:  4.8084e-01  3.5245e-01  9e-01  1e+00  2e-16  2e-01
 3:  8.4660e-01  7.8297e-01  2e-01  3e-01  2e-16  2e-02
 4:  8.9946e-01  8.7107e-01  9e-02  1e-01  6e-16  2e-03
 5:  9.2863e-01  9.1791e-01  3e-02  4e-02  7e-16  6e-04
 6:  9.4017e-01  9.3631e-01  1e-02  2e-02  1e-15  2e-04
 7:  9.4546e-01  9.4468e-01  2e-03  3e-03  2e-15  2e-05
 8:  9.4632e-01  9.4601e-01  1e-03  1e-03  1e-14  6e-06
 9:  9.4687e-01  9.4684e-01  1e-04  1e-04  2e-14  6e-07
10:  9.4692e-01  9.4692e-01  9e-06  1e-05  5e-15  4e-08
11:  9.4693e-01  9.4693e-01  9e-08  1e-07  1e-14  4e-10
12:  9.4693e-01  9.4693e-01  9e-10  1e-09  1e-14  4e-12
Optimal solution found.

----------------------------------------------------------------------------------------------------
--------

Results for lInfNorm though LP

----------------------------------------------------------------------------------------------------
--------

x= [-0.07646696 -0.02833069 -0.08875803 -0.04566501  0.00828258 -0.01264084
 -0.02596891 -0.03976349 -0.01617378  0.00940567]

---------------------------

A.dot(x) - b = [-0.93695265 -0.12128035 -0.13647516 ...  0.75428597  0.46370047
  0.0471121 ]
L1Norm(A.dot(x)-b) =  92.6214422686107
```

```
LInfNorm(A.dot(x)-b) =  0.9469281769410074
L2Norm(A.dot(x)-b) =  7.728424484060358

-----------------------------------------------------------------------------------------------
--------


-----------------------------------------------------------------------------------------------
--------

Results for lInfNorm though CVXPY

-----------------------------------------------------------------------------------------------
--------

x= [-0.07646696 -0.02833069 -0.08875803 -0.04566501  0.00828258 -0.01264084
 -0.02596891 -0.03976349 -0.01617378  0.00940567]

---------------------------

A.dot(x) - b = [-0.93695265 -0.12128035 -0.13647516 ...  0.75428597  0.46370047
  0.0471121 ]
L1Norm(A.dot(x)-b) =  92.6214422686107
LInfNorm(A.dot(x)-b) =  0.9469281769410074
L2Norm(A.dot(x)-b) =  7.728424484060359

-----------------------------------------------------------------------------------------------
--------

Solutions for the two methods for both L-Inf norma nd L-1 Norm are exactly the same
```

11. Least Squares Fitting

Implementation Notes

This was one of the hardest questions in the assignment as there were several parts to this question.

Challenges

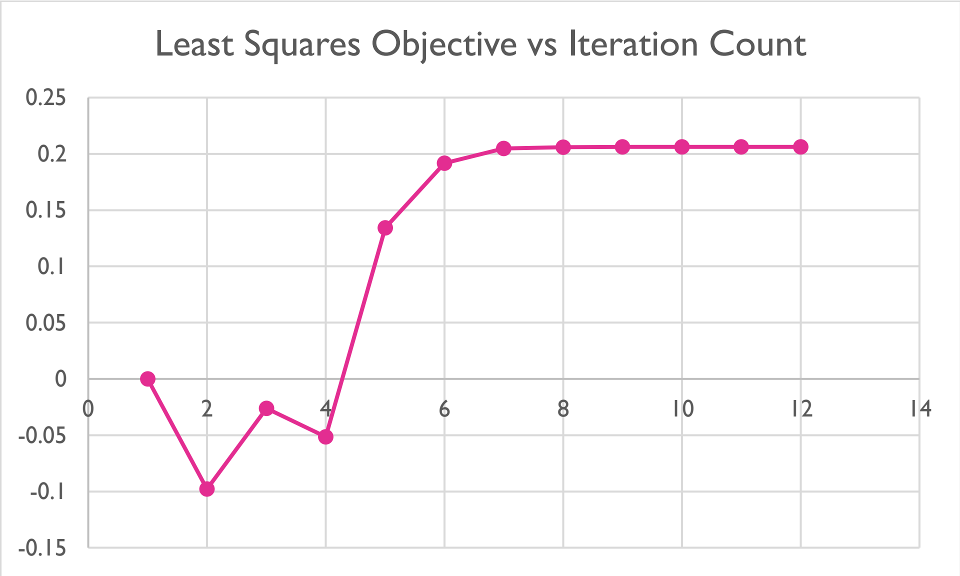
The main challenge for the first part was that there was no direct means to access the iteration counter. To get through this challenge, I wrote a simple text parser as below

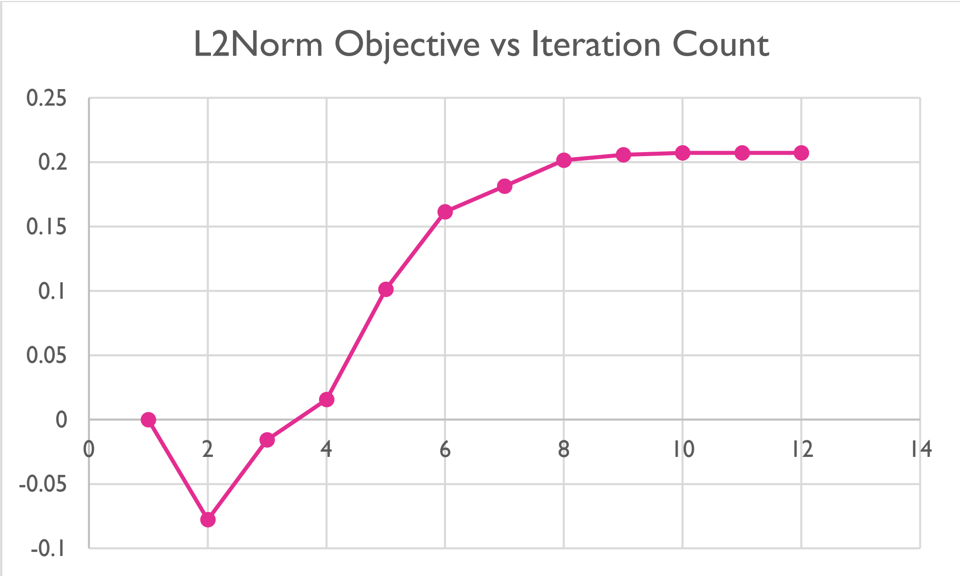
```
def getIterationErrors(problem, tempFile, column):
    originalOut = sys.stdout
    sys.stdout = open(tempFile, "w")
    a = problem.solve(verbose=True)
    sys.stdout.close()
    sys.stdout = originalOut
    f = open(tempFile, "r")
    linenum = 0
    iterationErrors = []
    for line in f:
        linenum += 1
        if linenum < 5:
            continue
        if line == "\n":
            break
        cols = line.split("  ")

        iterationErrors.append(float(cols[column]))
    return iterationErrors
```

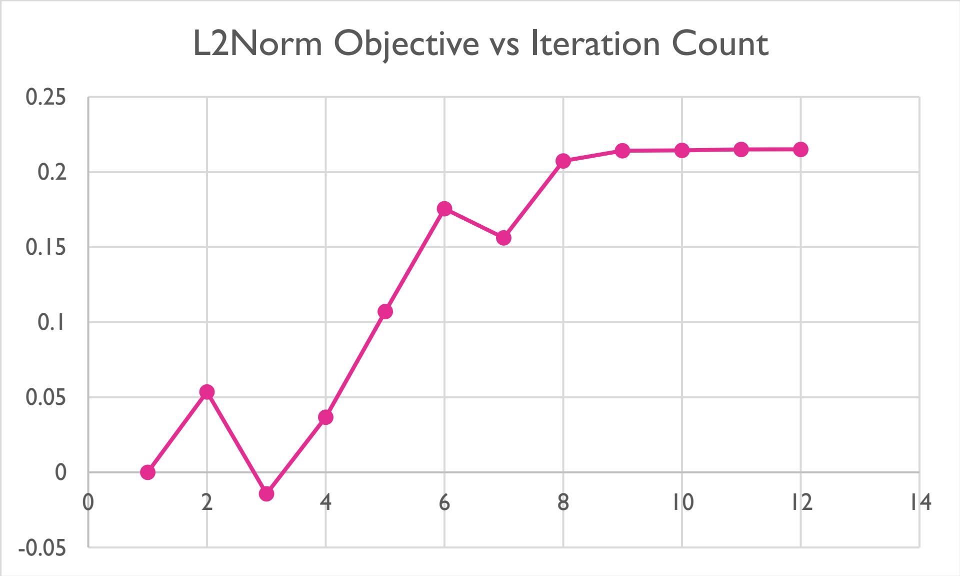
This pipes the output from the function from STDOUT into a file. From here, we can simply do a linewise parsing to obtain the column number required. In our case, we require the objective value in the second question and the error of the primal problem in the third question. We are able to get both, programmatically due to this simple parser.

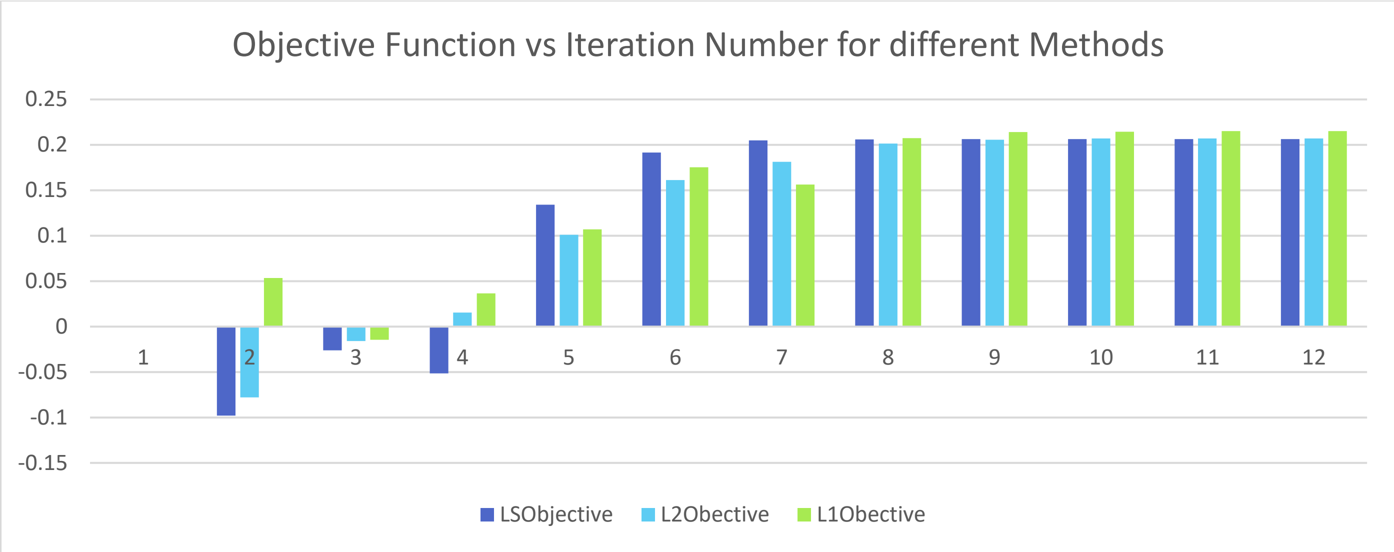
OUTPUTS

Part 1









Remarks

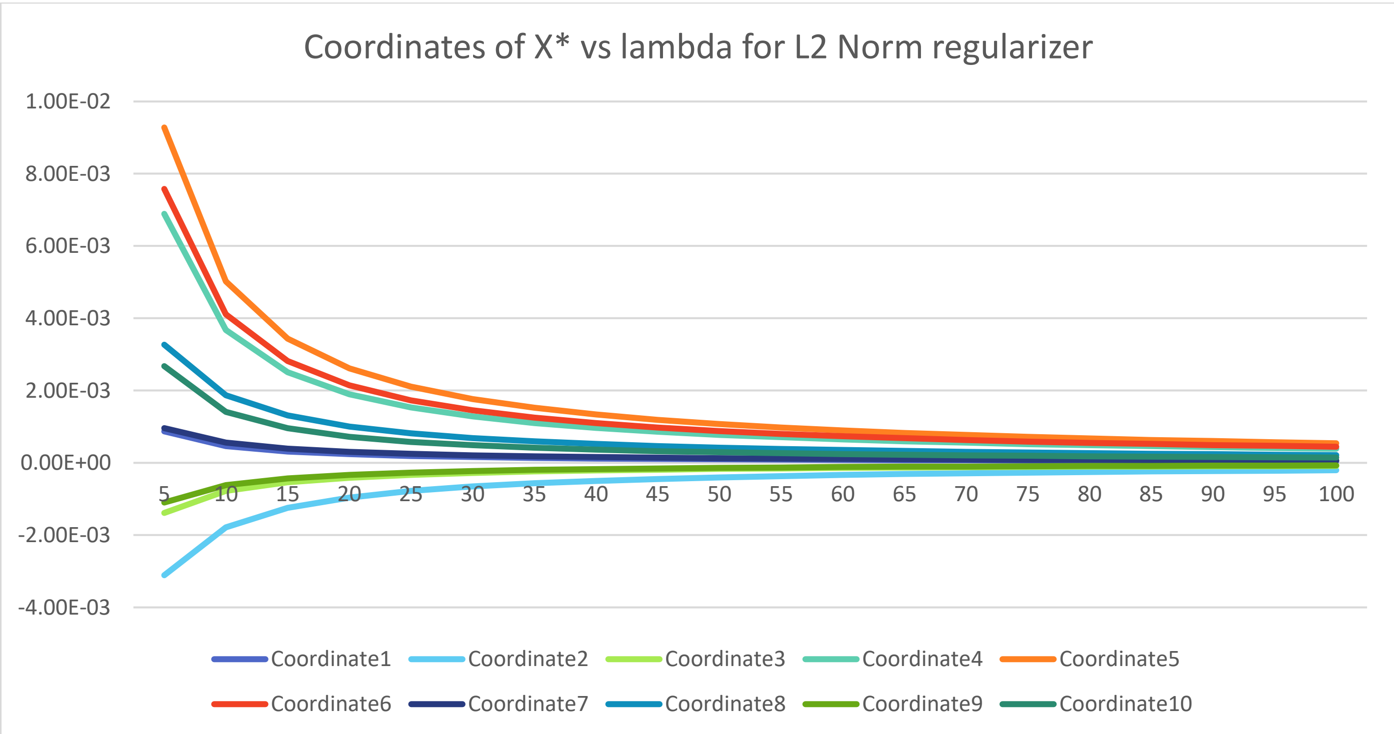
We notice that the above graphs show that the objective value for L1 is the largest. We shouldn't let this fool us into thinking that L1Norm doesn't work well. In fact it is the best alternative of the chosen methods. This is because the L1Norm tends to be the highest norm that the objective value is going slightly higher.

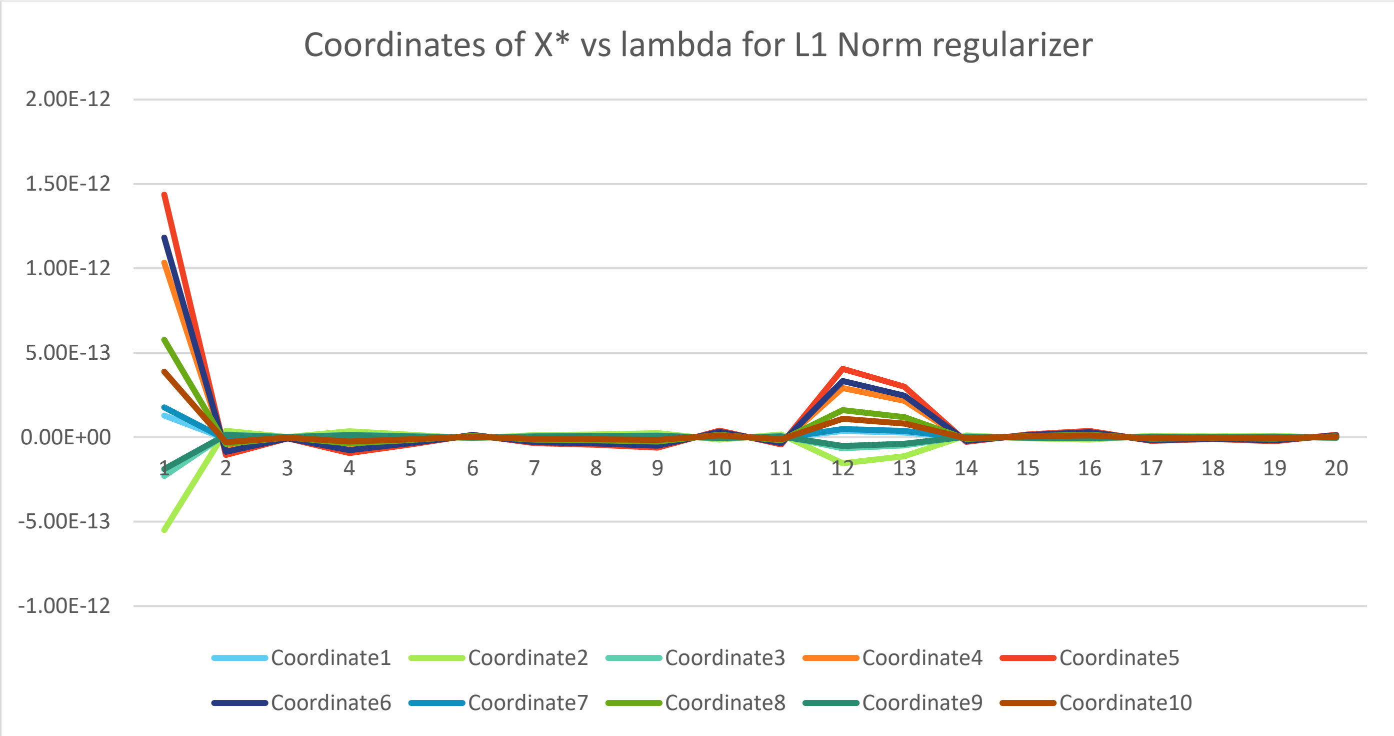
Otherwise all of the methods give the same value for the least squares part

Part 2

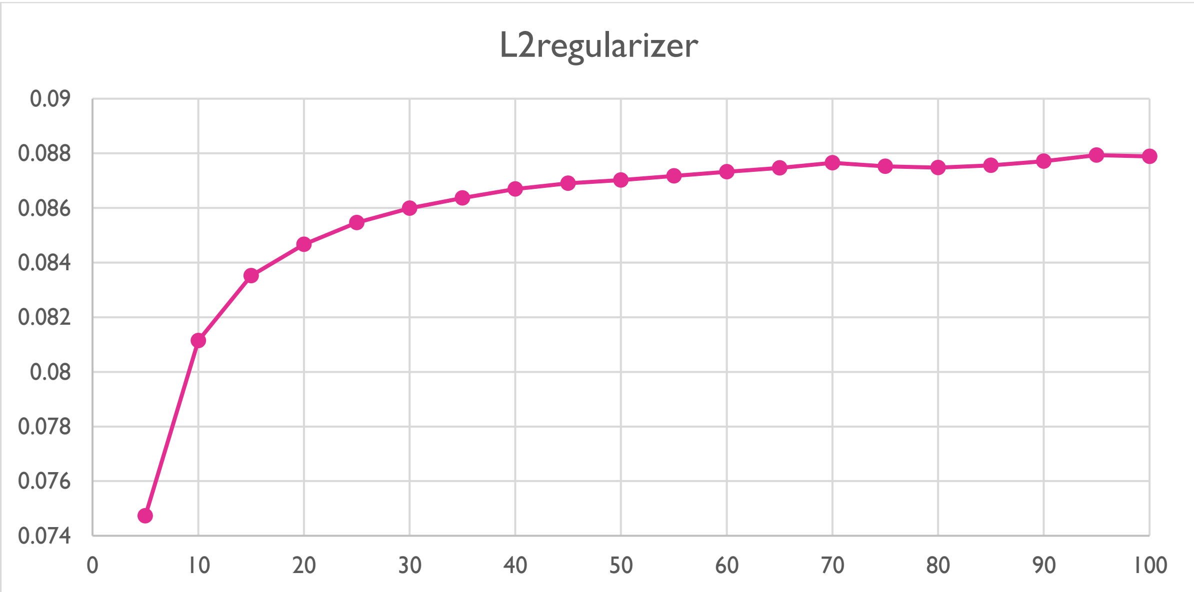
Here we see that the limits given in the question are extremely constraining, and thus we will expand the limits to see much better results.

Results with old values

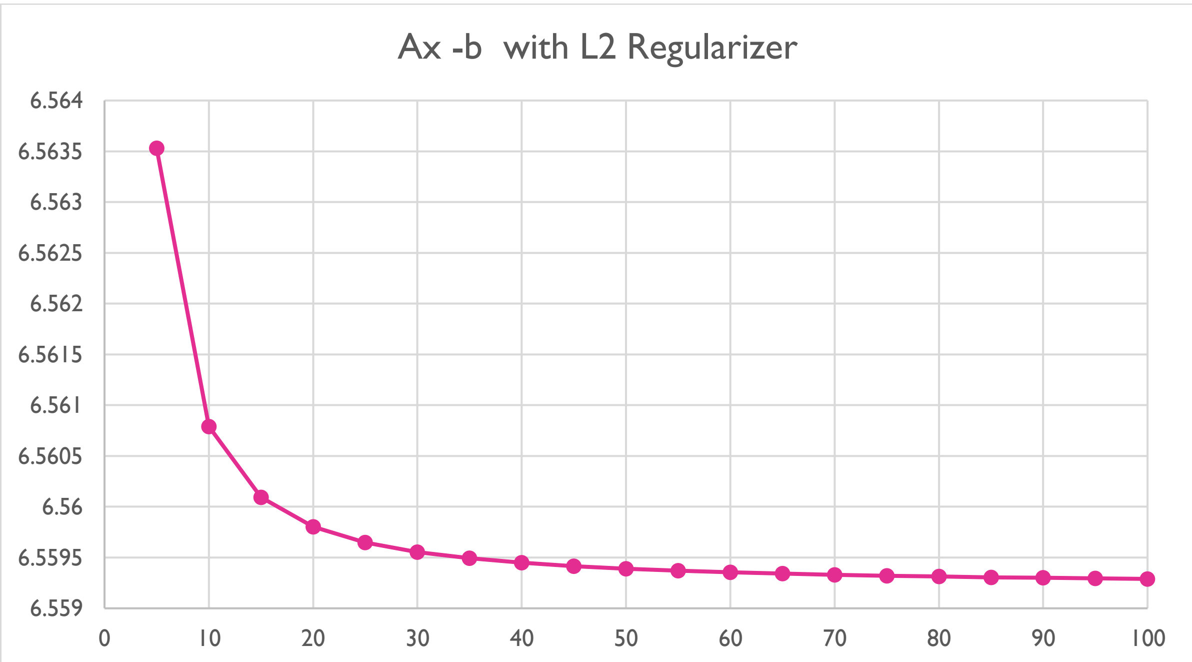




Coordinates of X* vs lambda for L1 Norm regularizer
2.00E-12
1.50E-12
1.00E-12
5.00E-13
0.00E+00
-5.00E-13
-1.00E-12
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Coordinate1
Coordinate2
Coordinate3
Coordinate4
Coordinate5
Coordinate6
Coordinate7
Coordinate8
Coordinate9
Coordinate10

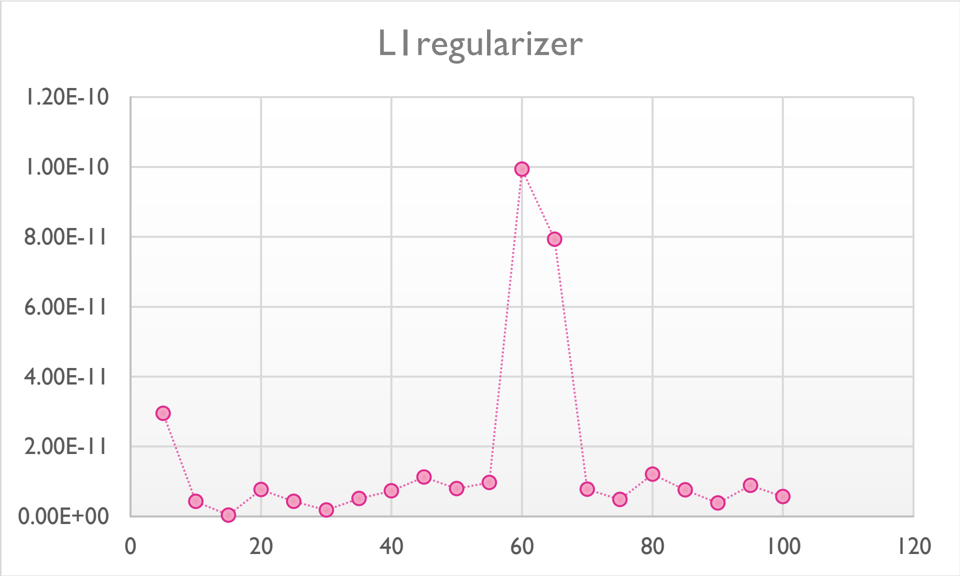


L2regularizer
0.09
0.088
0.086
0.084
0.082
0.08
0.078
0.076
0.074
0 10 20 30 40 50 60 70 80 90 100

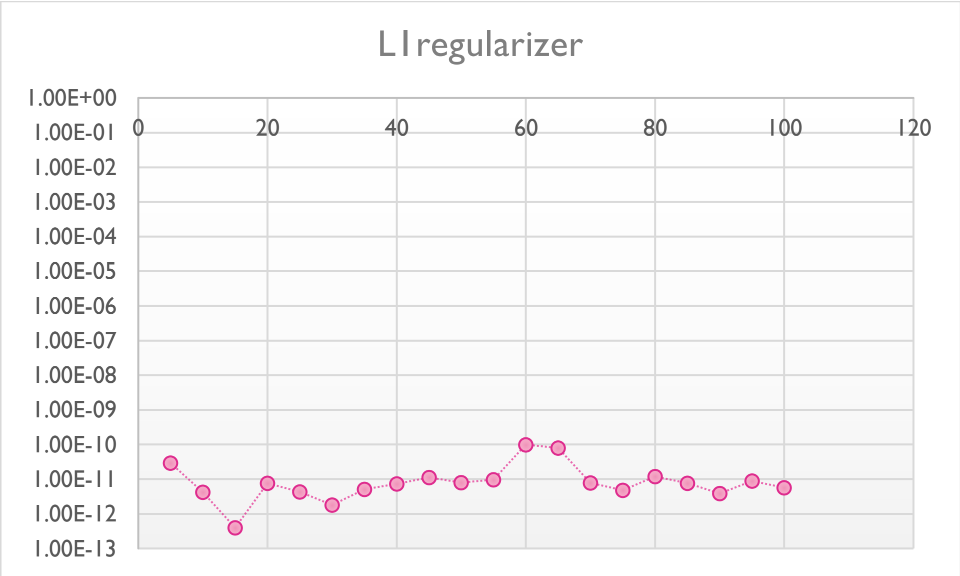


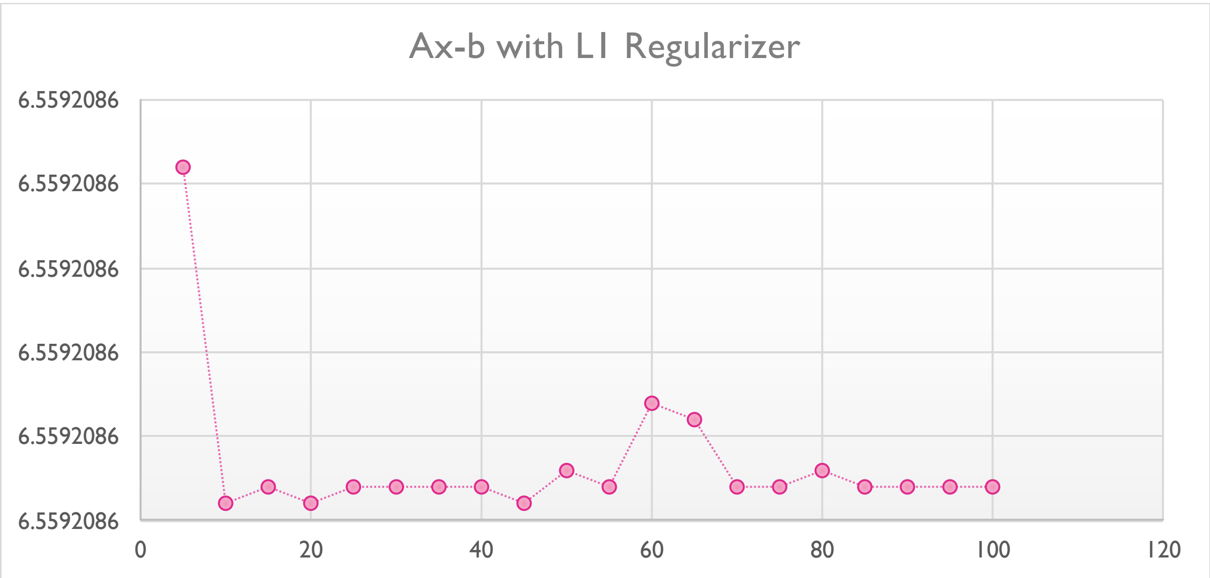
Ax -b with L2 Regularizer
6.564
6.5635
6.563
6.5625
6.562
6.5615
6.561
6.5605
6.56
6.5595
6.559
0 10 20 30 40 50 60 70 80 90 100

Actuals: L1 Regularizer



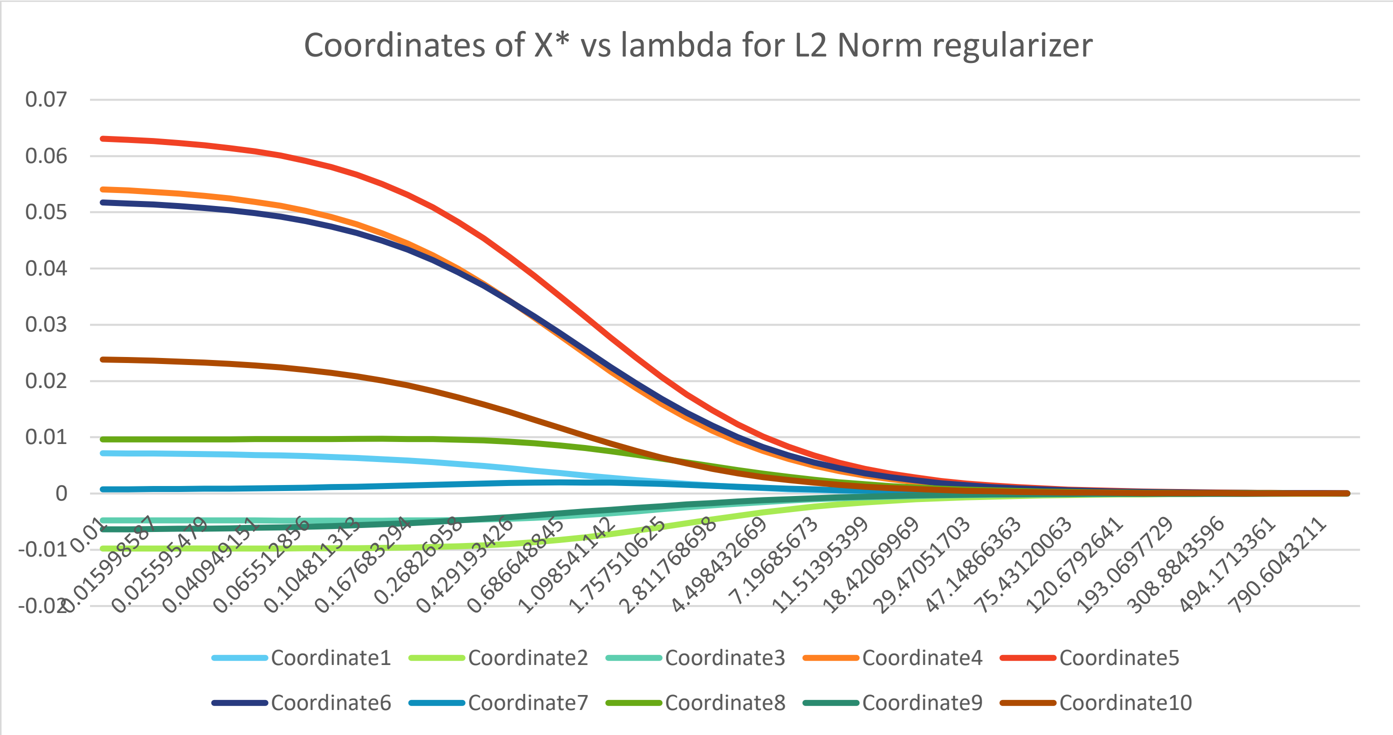
Vs Log Scale: L1 Regularizer

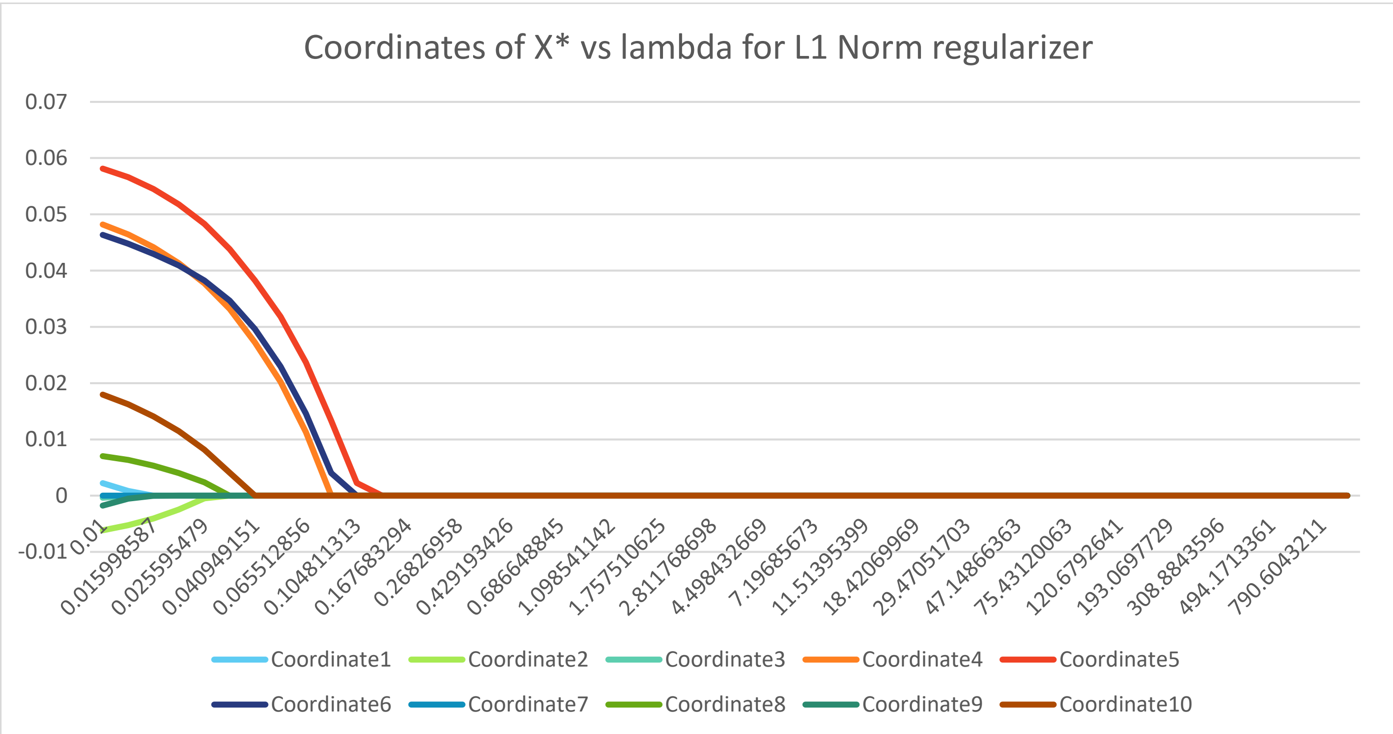




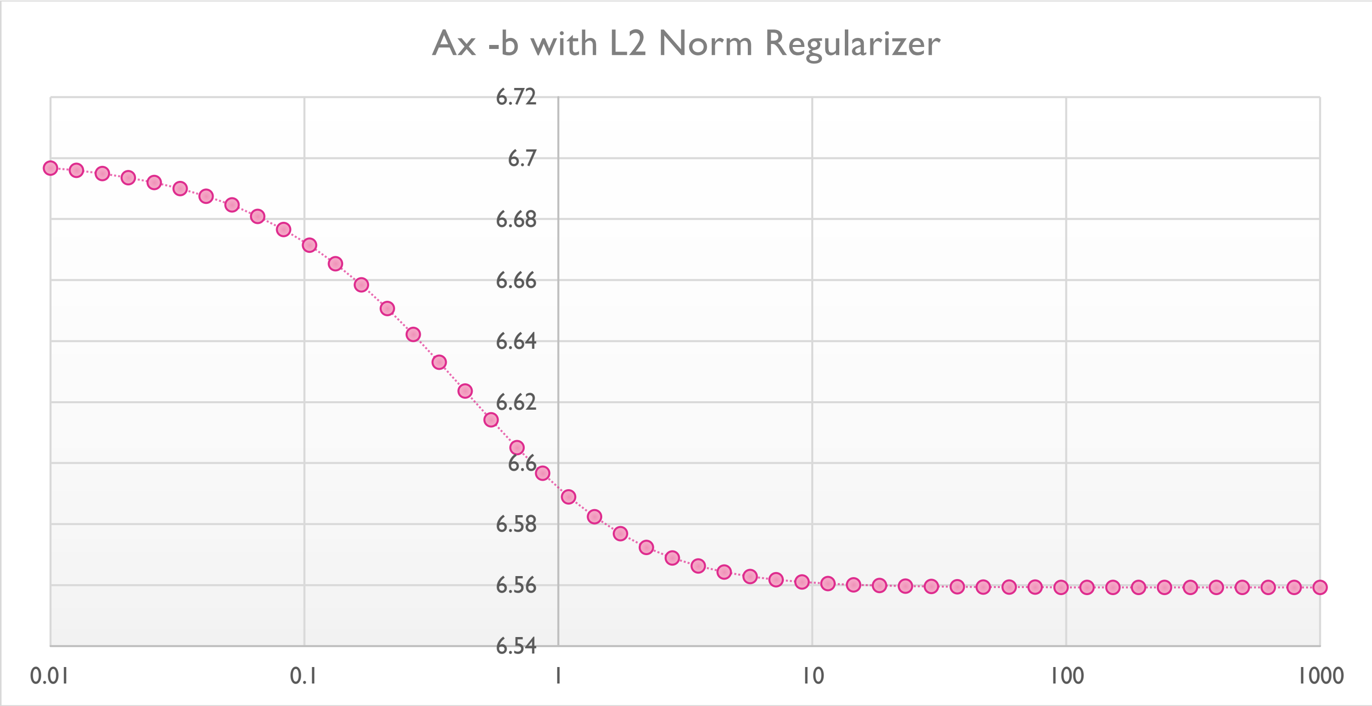
As one can notice, the above curves are extremely jagged especially for L1 norm regularization. **This called for further investigation** as the curve did not make too much sense

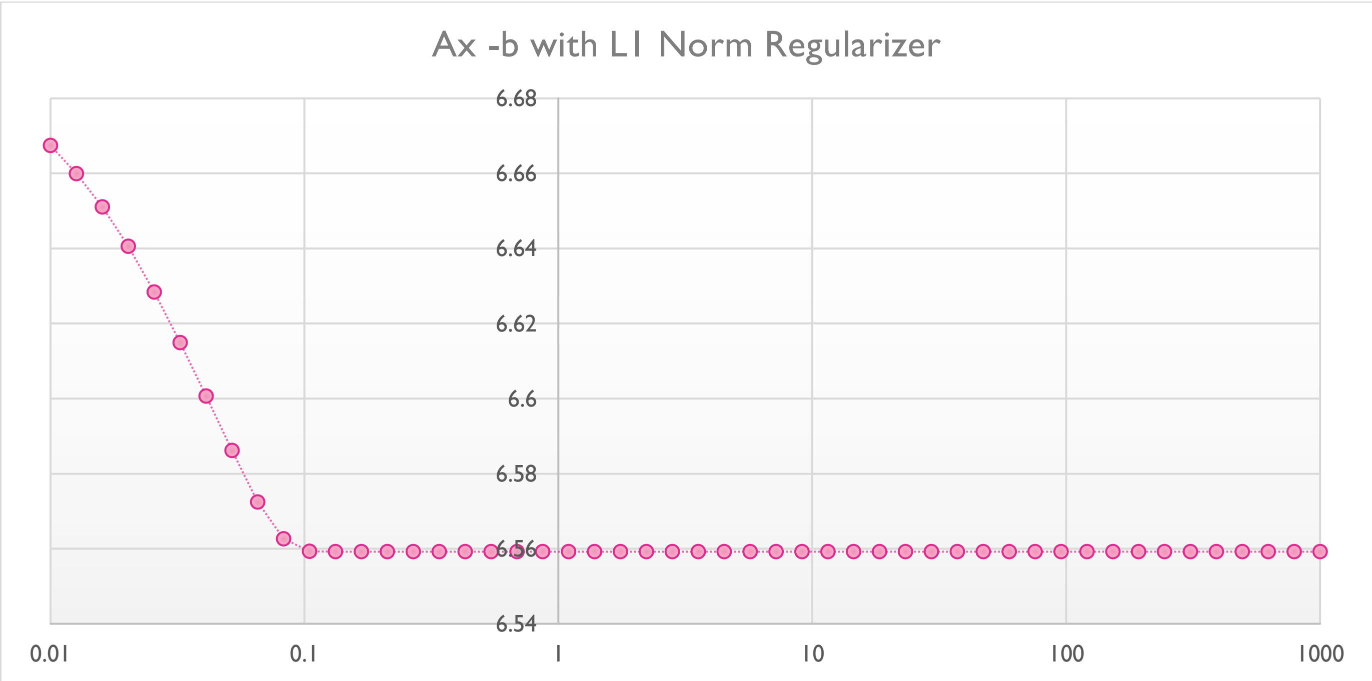
Therefore we plot a much wider range of lambda values below which gives good results



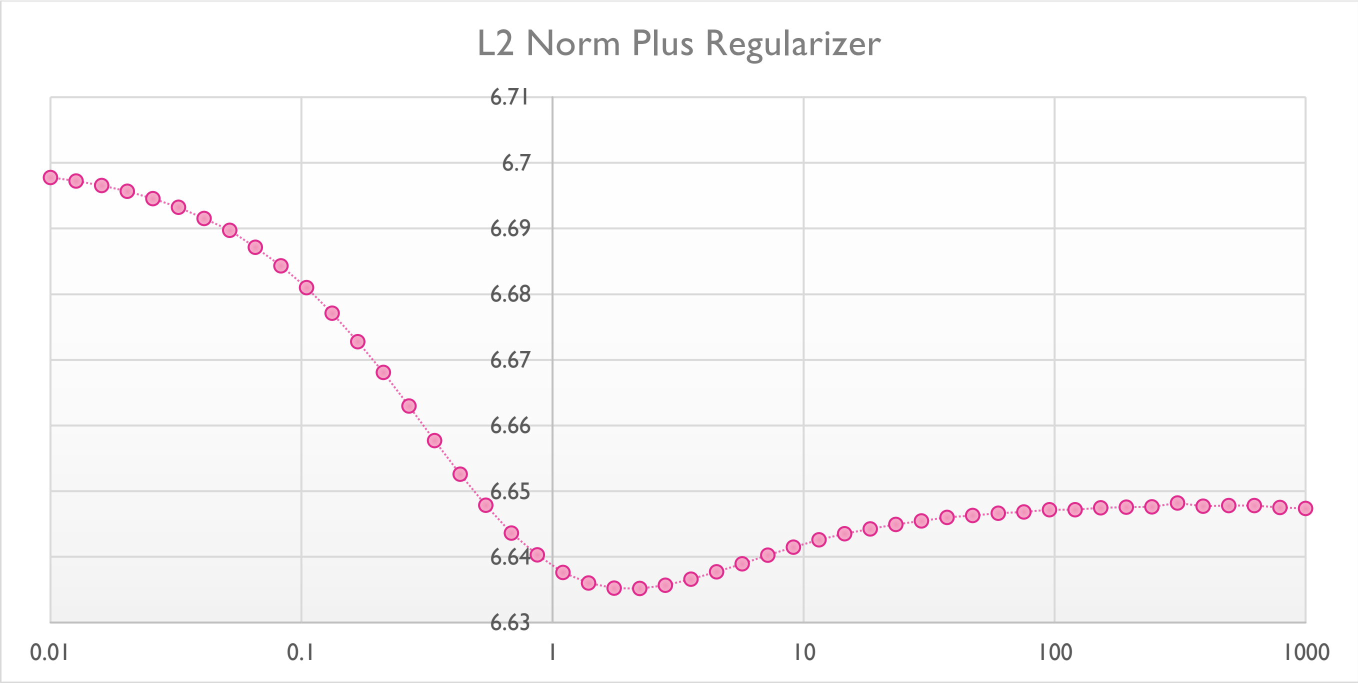


Notice in the above graph how nicely we see the various coordinates going to zero. These actually go to zero much below the lowest earlier value investigated, that is 5.

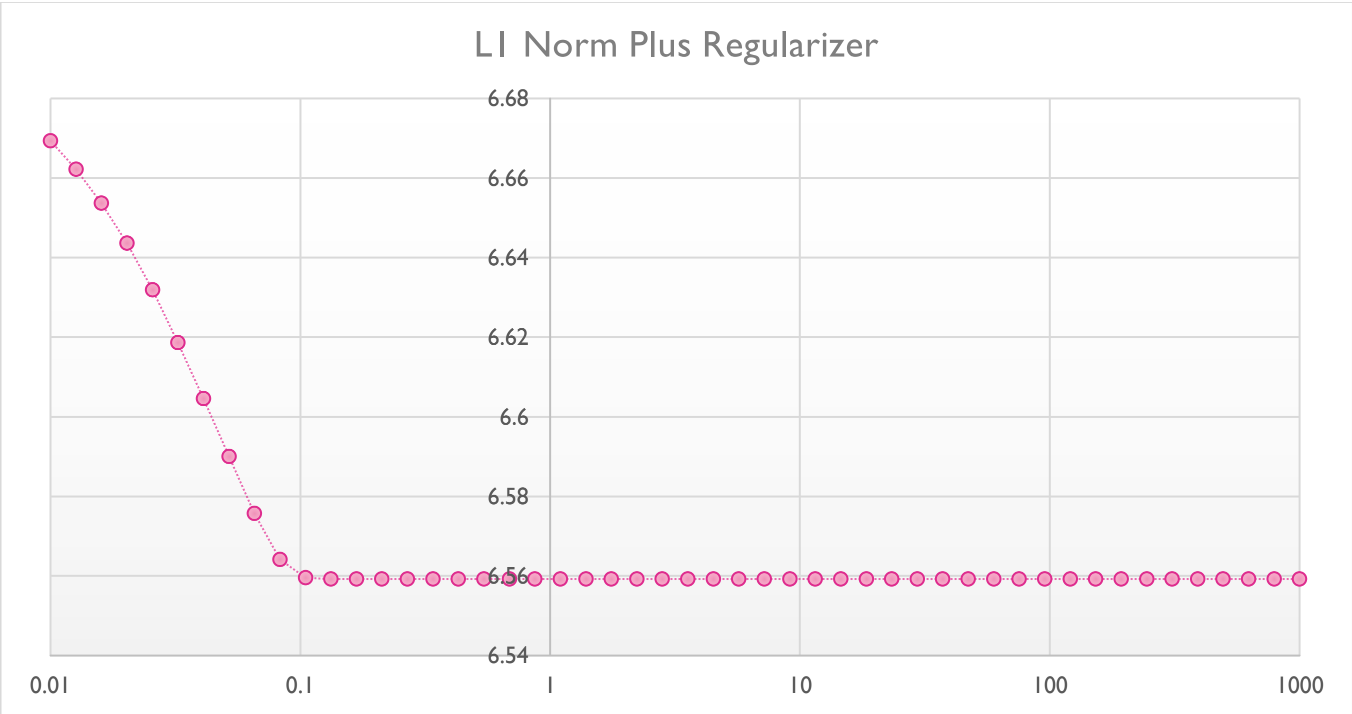




Notice that our objective function for L2 norm goes to zero as early as around 2.25 and for L1 norm at around 0.1



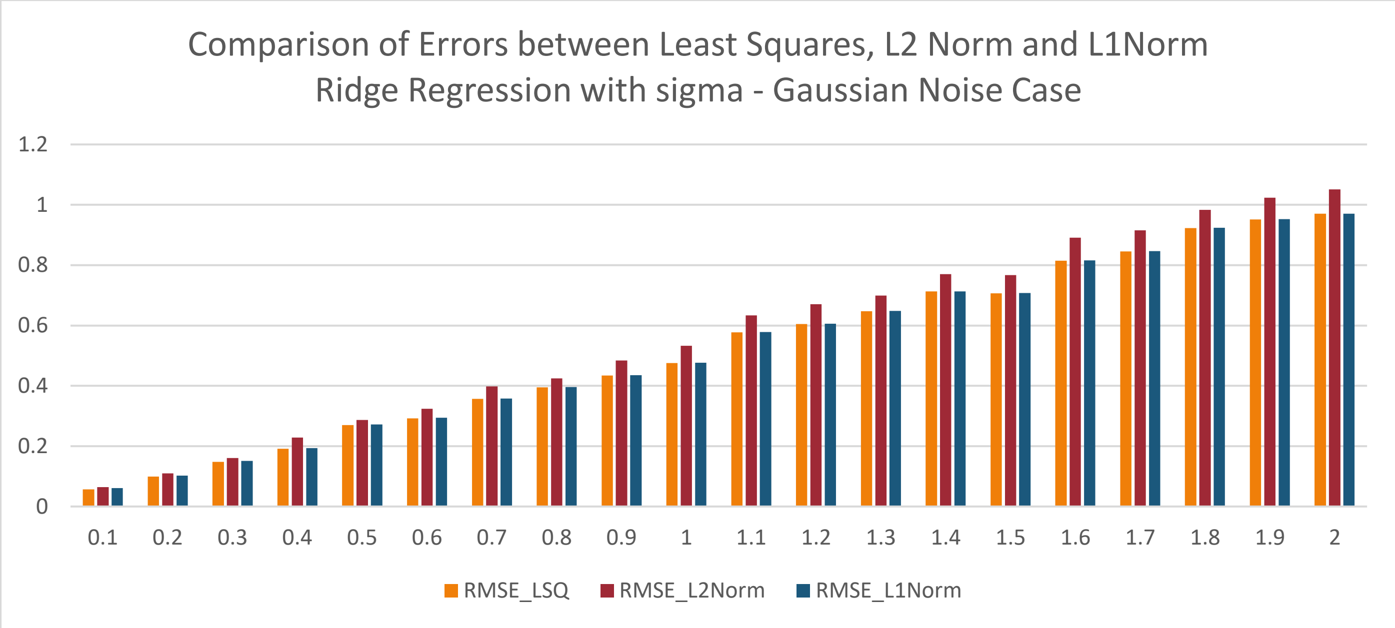
We plot the norm plus regularizer to see the results more clearly. The inflection in our data was at around 2.25 lambda value



Similarly, the inflection point for L1 is seen at around 0.1

Part 3

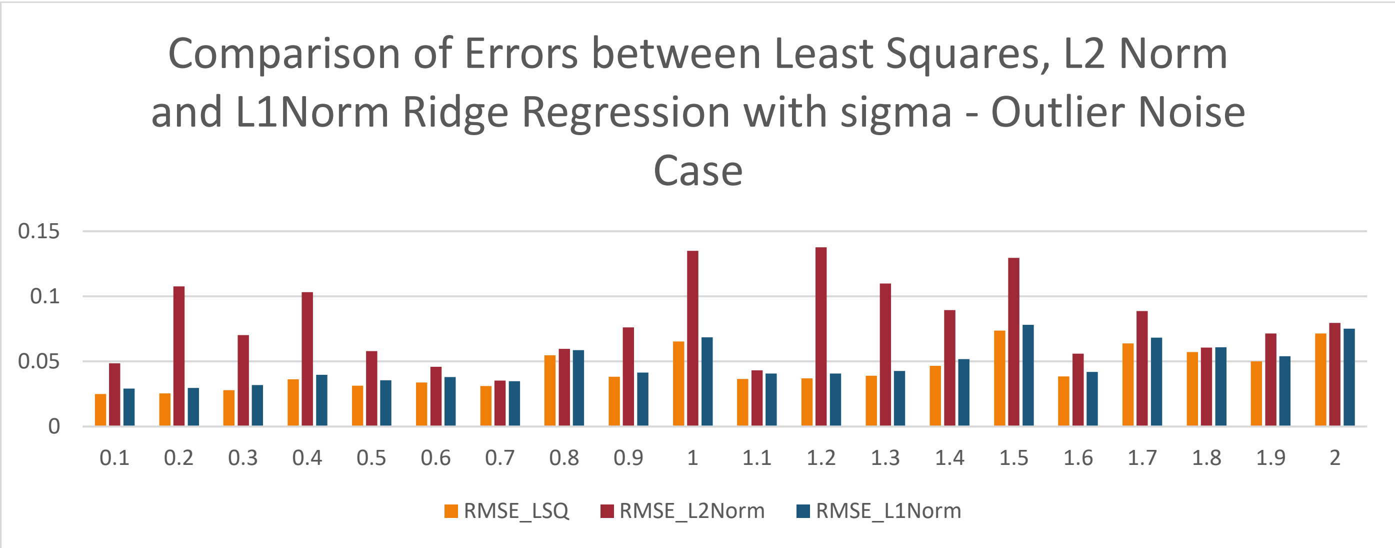
We now plot the RMSE values vs sigma for various values of sigma chosen. Note that we use lambda = 0.1 for L1 Reg and lambda = 2.25 for L2 Reg



Notice that the error values uniformly go up with higher sigma values. Notice that L1 norm regularizer performs the best and L2 doesn't really add any value

Part 4

Here, instead of Gaussian Noise, we add a individual noise component



Again, we notice that L2 norm regularizer indeed performs the worst.

III SVM Fitting

Remarks

- **We were able to find the error function per iteration, even though clarification was issued that this is not required**
- Problem solved for different extended parameters even apart from those asked
- Proof of correctness provided in the following page
- We provide plenty of graphs for ease of understanding of implementation
- All code provided in the appendix
- Works only with certain solvers like CVXOPT
- Some of the results of the digits separation were surprising. In some cases, similar looking digits were classified well and very different digits were not that well classified
- We got better results from the regular solver than the Gaussian Kernel
- Training Data is 0 as we need linear separation in second part

Challenges

This problem along with the second problem is one of the hardest problems of the entire assignment. This was particularly challenging due to unexplained failures of solvers in DCP (disciplined convex programming). Finally the error did not get fixed on a local machine but was fixed when one ran the same code on Google Collab.

Part 1 Proof of Correctness

This problem along with the second problem is one of the hardest problems of the entire assignment. This was particularly challenging due to unexplained failures of solvers in DCP (disciplined convex programming). Finally the error did not get fixed on a local machine but was fixed when one ran the same code on Google Collab.

PRIMAL | Minimize $\frac{1}{2} w^T w$

Subject To $1 - y_i (w^T x_j + b) \le 0$

$j = 1 \ldots m$

<u>Problem</u>: Find the Dual Problem

<u>Answer</u>:

The primal constraint is

$$1 - y_i (w^T x_j + b) \le 0 \quad \forall j = 1 \ldots m$$

This can be written as

$$\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} - \left[- y_i - \right] \left[\begin{bmatrix} - x_1 - \\ - x_2 - \\ \vdots \\ - x_m - \end{bmatrix} \begin{bmatrix} | \\ w \\ | \end{bmatrix} - \begin{bmatrix} b \\ b \\ b \\ \vdots \\ b \end{bmatrix} \right] \le 0$$

OR $\quad 1 - Y^T (X^T w - b \cdot 1) \le 0 \quad$ where $1 \in \mathbb{R}^m$

Then $\mathcal{L}(w, b, \lambda) = \frac{1}{2} w^T w + \lambda^T (1 - Y^T (X^T w - b \cdot 1))$ — (1)

is the Lagrangian $\boxed{\text{where } \lambda \ge 0}$ — (1a)

Setting Derivatives with respect to w & b equal to 0.

$$\frac{\partial L}{\partial W} = 0 \Rightarrow \frac{\partial}{\partial W}\left(\frac{W^T W}{2} + \lambda^T\left(1 - Y^T(X^T W - b.1)\right)\right) = 0$$

$$\Rightarrow W - XY\lambda = 0$$

$$\Rightarrow W = XY\lambda \quad \text{— (2)}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \frac{\partial}{\partial b}\left(\frac{w^T w}{2} + \lambda^T\left(1 - Y^T(X^T w - b.1)\right)\right) = 0$$

$$\Rightarrow \frac{\partial}{\partial b}\left(+ \lambda^T Y^T b.\ 1\right) = 0$$

$$\Rightarrow \lambda^T Y = 0 \quad \text{— (3)}$$

We can substitute these values in (1)

$$\rightarrow \mathcal{L}(\lambda, w, b) = \frac{1}{2} w^T w + \lambda^T 1 - \lambda^T Y^T X^T W + b.\lambda^T Y^T 1$$

From (3) $$\mathcal{L}(w, b, \lambda) = \frac{1}{2} w^T w + \lambda^T 1 - \lambda^T Y^T X^T w + 0$$

From (2) $$\mathcal{L}(w, b, \lambda) = \lambda^T 1 + \frac{1}{2} w^T w - w^T w$$

$$= \lambda^T 1 - \frac{1}{2} w^T w$$

But $w = X Y \lambda$

Then

$$L(w, b, \lambda) = \lambda^T 1 - \frac{1}{2} \lambda^T Y^T X^T X Y \lambda$$

Let $X^T X$ be called Σ

Then $L(w, b, \lambda) = \lambda^T 1 - \frac{1}{2} \lambda^T (Y^T \Sigma Y) \lambda$

But because of 3 we have the condition

$$\lambda^T y = 0$$

Further, because of 1a we have

$$\lambda \geq 0.$$

Since we are trying to maximize this Lagrangian, we can state that the dual problem is

$$\text{Maximize} \quad \lambda^T 1 - \frac{1}{2} \lambda^T (Y^T \Sigma Y) \lambda$$

$$\text{Subject to} \quad \lambda \geq 0$$

$$y^T \lambda = 0$$

where $1 \in \mathbb{R}^m$

Now ~~we see~~ for the second part we can see that the Slaters Conditions hold because the constraints are linear in w

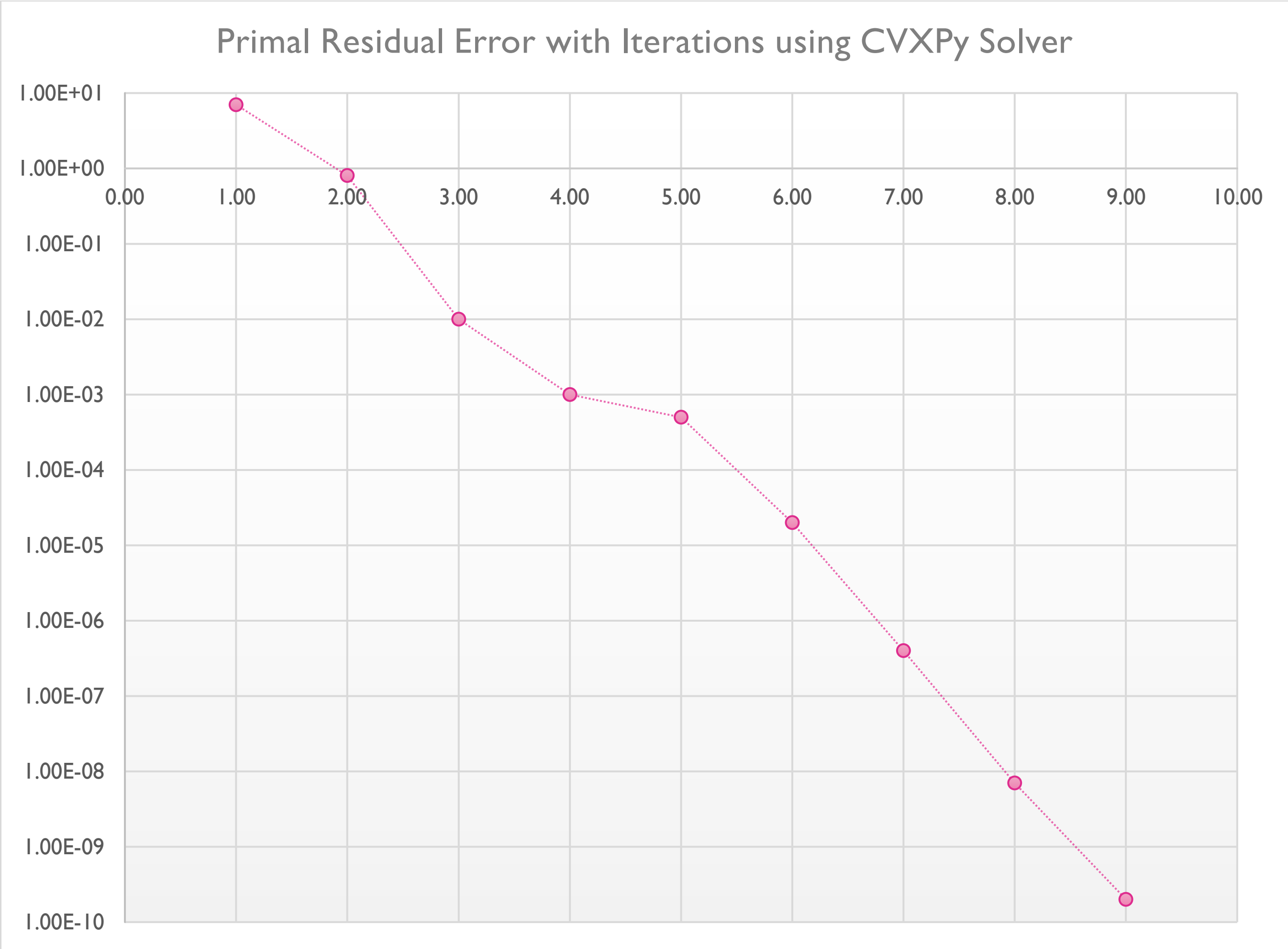
Thus, $\exists$ points in w such that strong inequality holds.

Thus we see that strong duality holds.

Thus $w = XY\lambda^*$ for a λ^* solution of Dual.

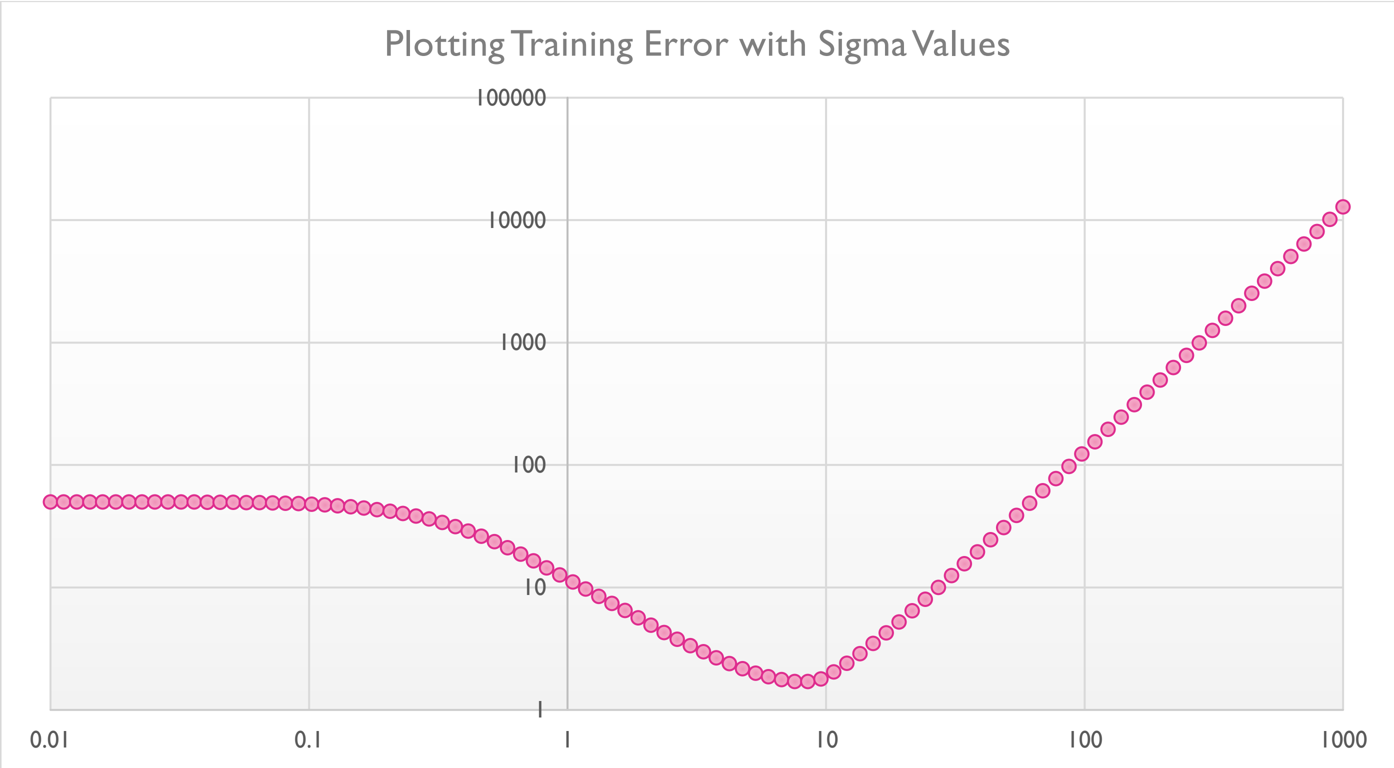
Thus $\boxed{w = \sum_{i=1}^{m} \lambda_i^* y_i x_i}$ as required

Part 2

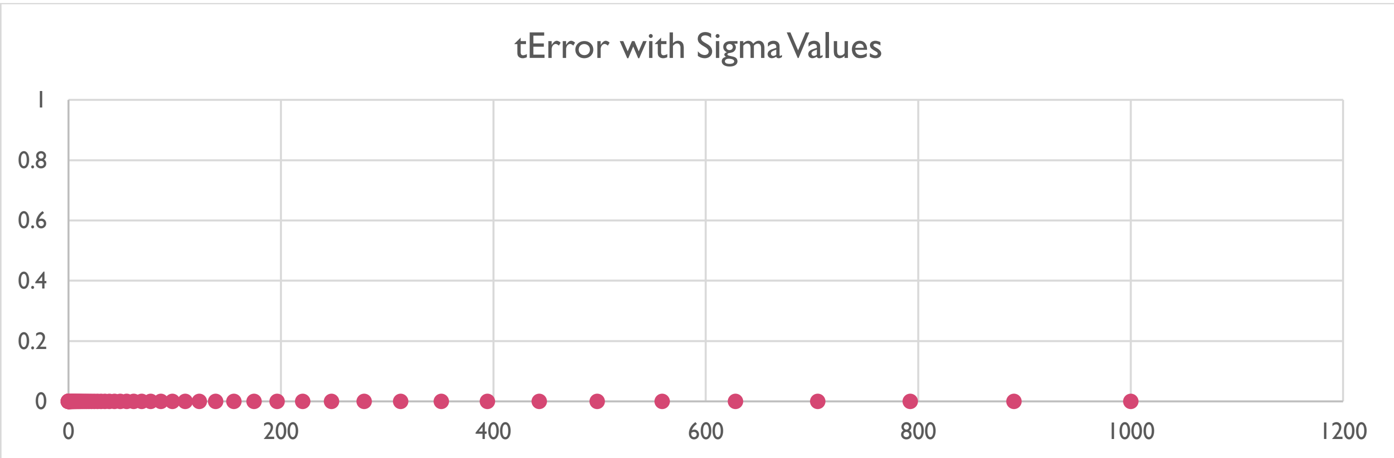


Here we have been able to plot the training error per iteration, even though this was cancelled in the original question

Part 3

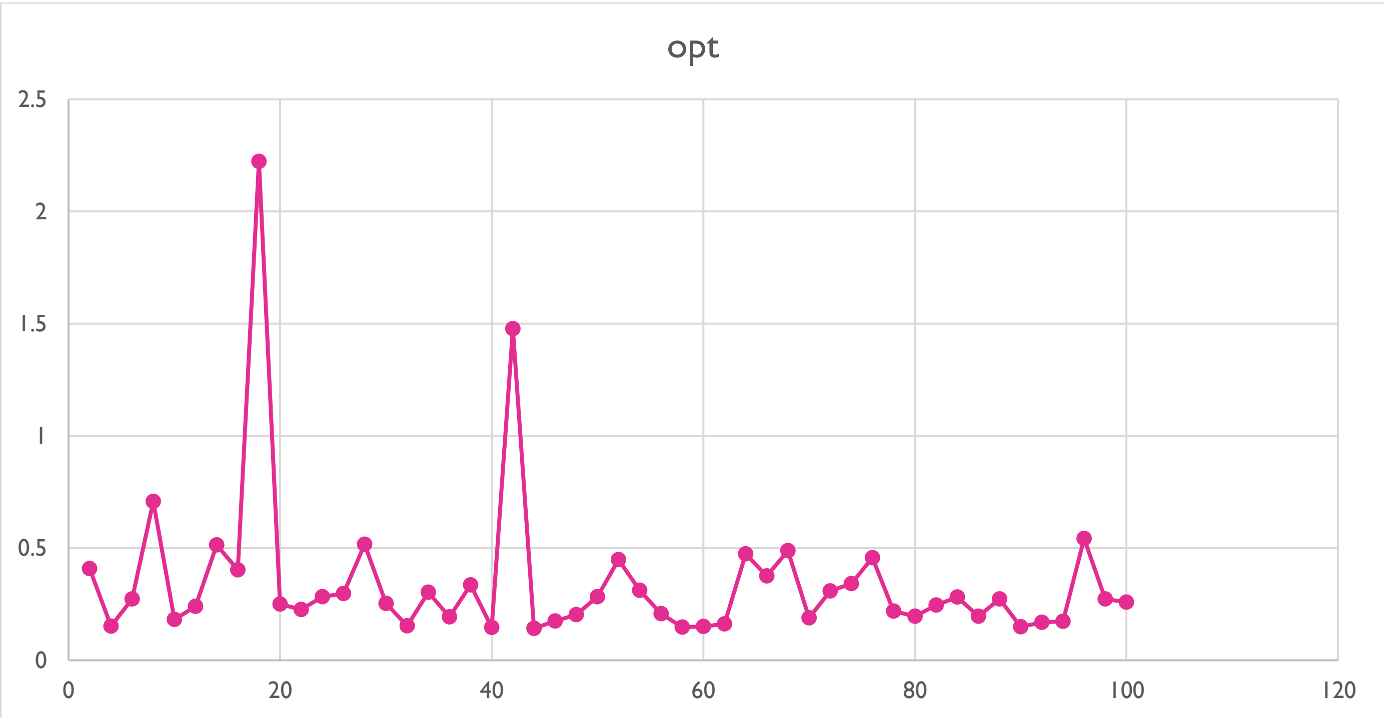


Plotting Training Error with Sigma Values
100000
10000
1000
100
10
1
0.01
0.1
1
10
100
1000



tError with Sigma Values
1
0.8
0.6
0.4
0.2
0
0
200
400
600
800
1000
1200

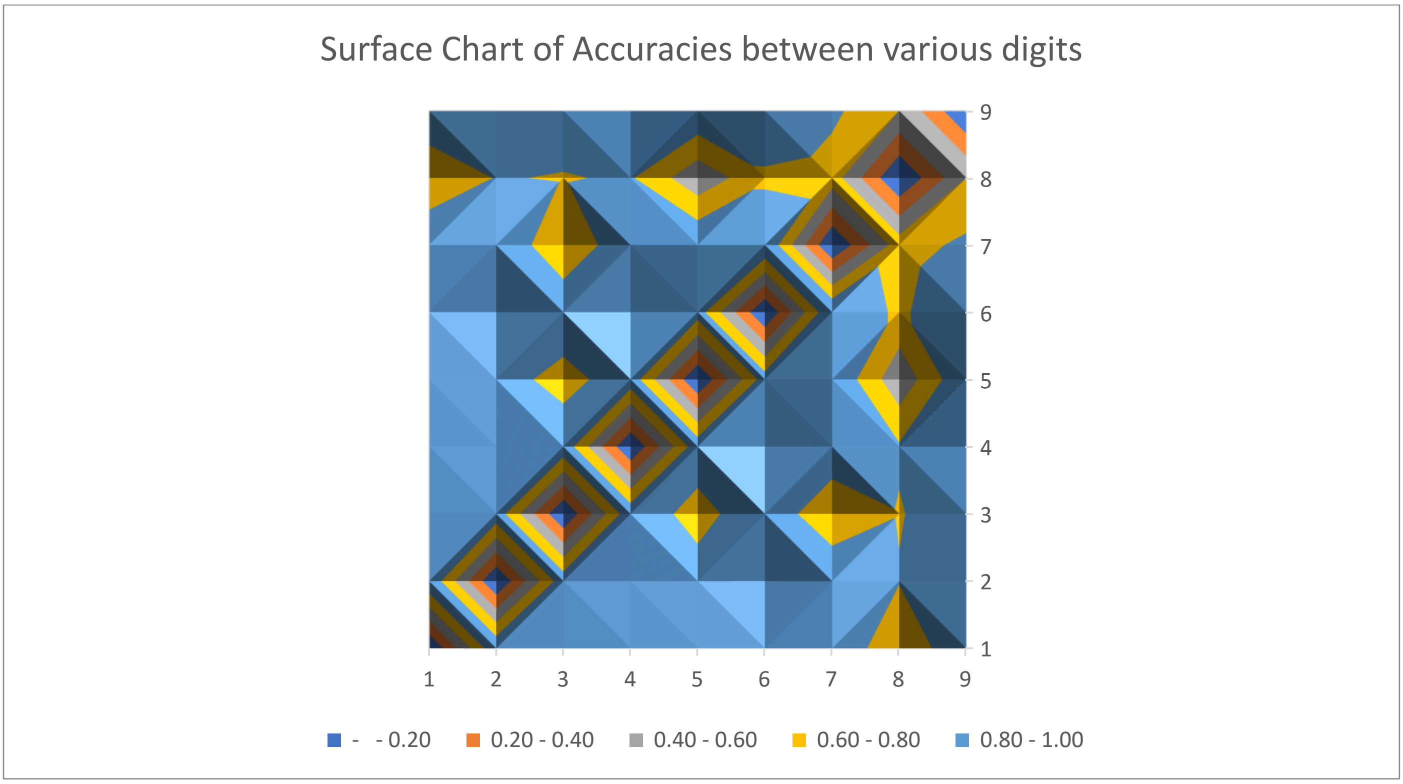
distance\lambda	2.5	3	3.5	4	4.4
2	0.13	0.12	0.21	0.09	0.10
4	0.11	0.14	0.09	0.11	0.09
6	0.07	0.12	0.12	0.41	0.15
8	0.10	0.14	0.12	0.14	0.12
10	0.16	0.09	0.16	0.09	0.11
12	0.10	0.11	0.14	0.10	0.13
14	0.15	0.09	0.11	0.10	0.12
16	0.12	0.08	0.12	0.14	0.08
18	0.14	0.16	0.14	0.28	0.21
20	0.08	0.09	0.14	0.10	0.13
30	0.07	0.20	0.10	0.15	0.10
40	0.10	0.09	0.10	0.12	0.12
50	0.12	0.08	0.16	0.11	0.12
60	0.18	0.20	0.15	0.08	0.10
70	0.09	0.20	0.15	0.12	0.09
80	0.07	0.12	0.12	0.13	0.14
90	0.13	0.12	0.23	0.09	0.12
100	0.10	0.12	0.13	0.13	0.13

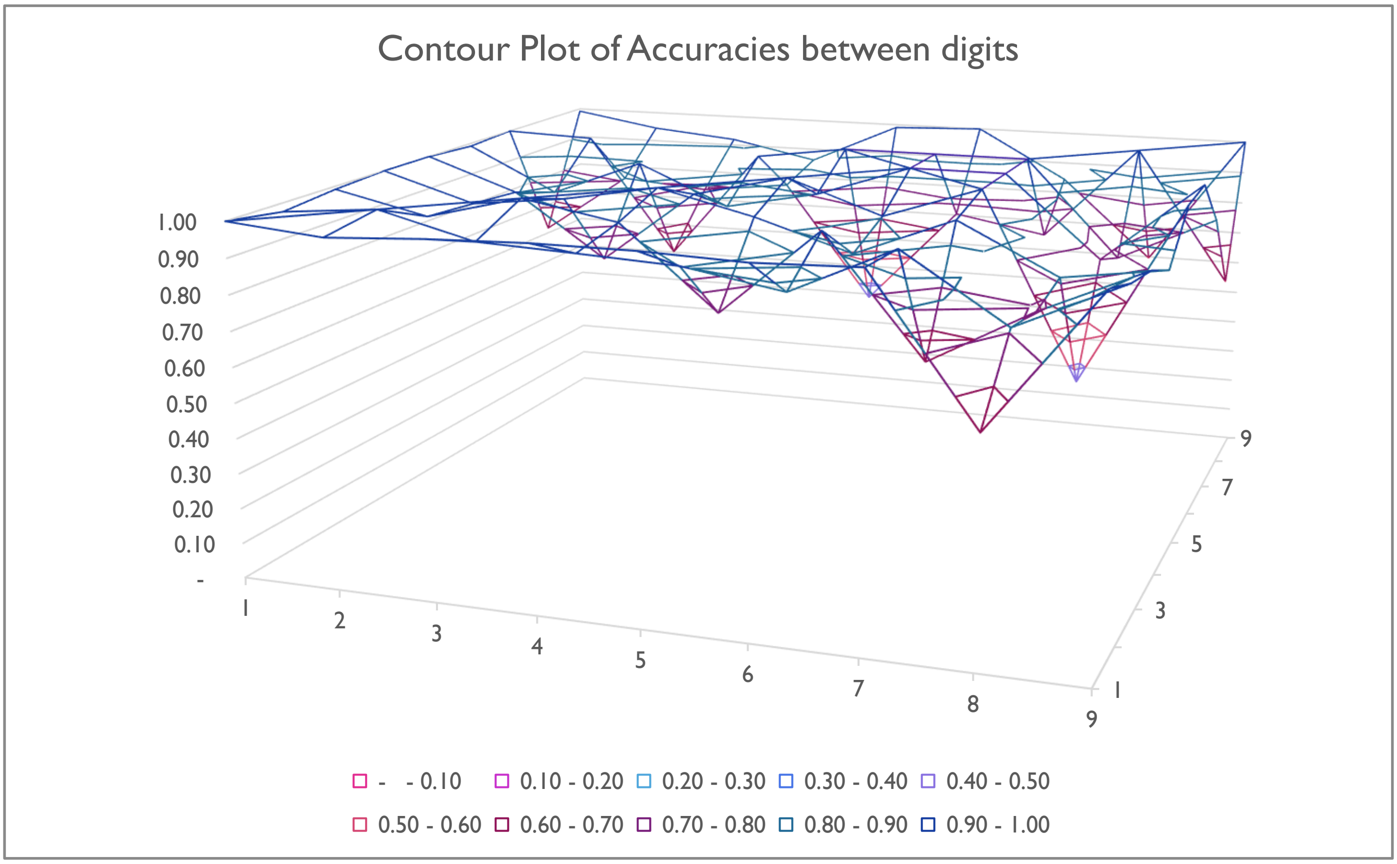


The above graph is quite arbitrary and indicates no major dependence

Part 4

We now indicate the test errors between various digits under consideration. Notice that these have been given as surface plots, where the surface colour indicates the error for that pair. The graphs are clearly symmetric





D1\D2	1	2	3	4	5	6	7	8	9
1		0.98	0.99	1.00	1.00	0.99	1.00	0.62	0.99
2	0.98		0.93	0.92	0.90	0.86	0.99	0.81	0.94
3	0.99	0.93		0.95	0.72	0.97	0.63	0.79	0.92
4	1.00	0.92	0.95		0.93	0.85	0.96	0.82	0.86
5	1.00	0.90	0.72	0.93		0.90	1.00	0.46	0.99
6	0.99	0.86	0.97	0.85	0.90		1.00	0.76	1.00
7	1.00	0.99	0.63	0.96	1.00	1.00		0.71	0.84
8	0.62	0.81	0.79	0.82	0.46	0.76	0.71		0.60
9	0.99	0.94	0.92	0.86	0.99	1.00	0.84	0.60	

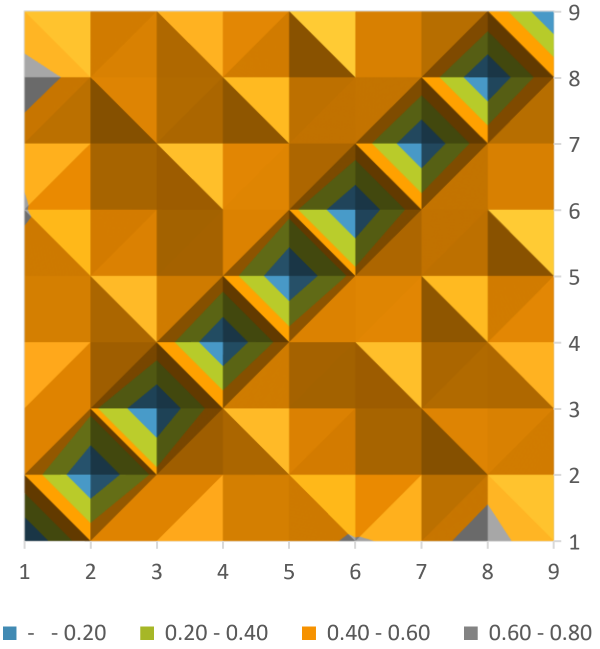
Accuracies using Gaussian Kernels

We see that we get much lower accuracy values using Gaussian Kernels. The reason for this is not immediately apparent

On average, the accuracies here are lower by a factor of around 0.2

D1\D2	1	2	3	4	5	6	7	8	9
1		0.55	0.60	0.55	0.57	0.61	0.58	0.63	0.55
2	0.55		0.45	0.50	0.52	0.44	0.47	0.58	0.50
3	0.60	0.45		0.55	0.47	0.49	0.52	0.53	0.55
4	0.55	0.50	0.55		0.52	0.56	0.47	0.58	0.50
5	0.57	0.52	0.47	0.52		0.46	0.49	0.44	0.52
6	0.61	0.44	0.49	0.56	0.46		0.53	0.52	0.44
7	0.58	0.47	0.52	0.47	0.49	0.53		0.55	0.48
8	0.63	0.58	0.53	0.58	0.44	0.52	0.55		0.57
9	0.55	0.50	0.55	0.50	0.52	0.44	0.48	0.57	

Surface Chart of Accuracies between various digits with Gaussian Kernel



Question 4

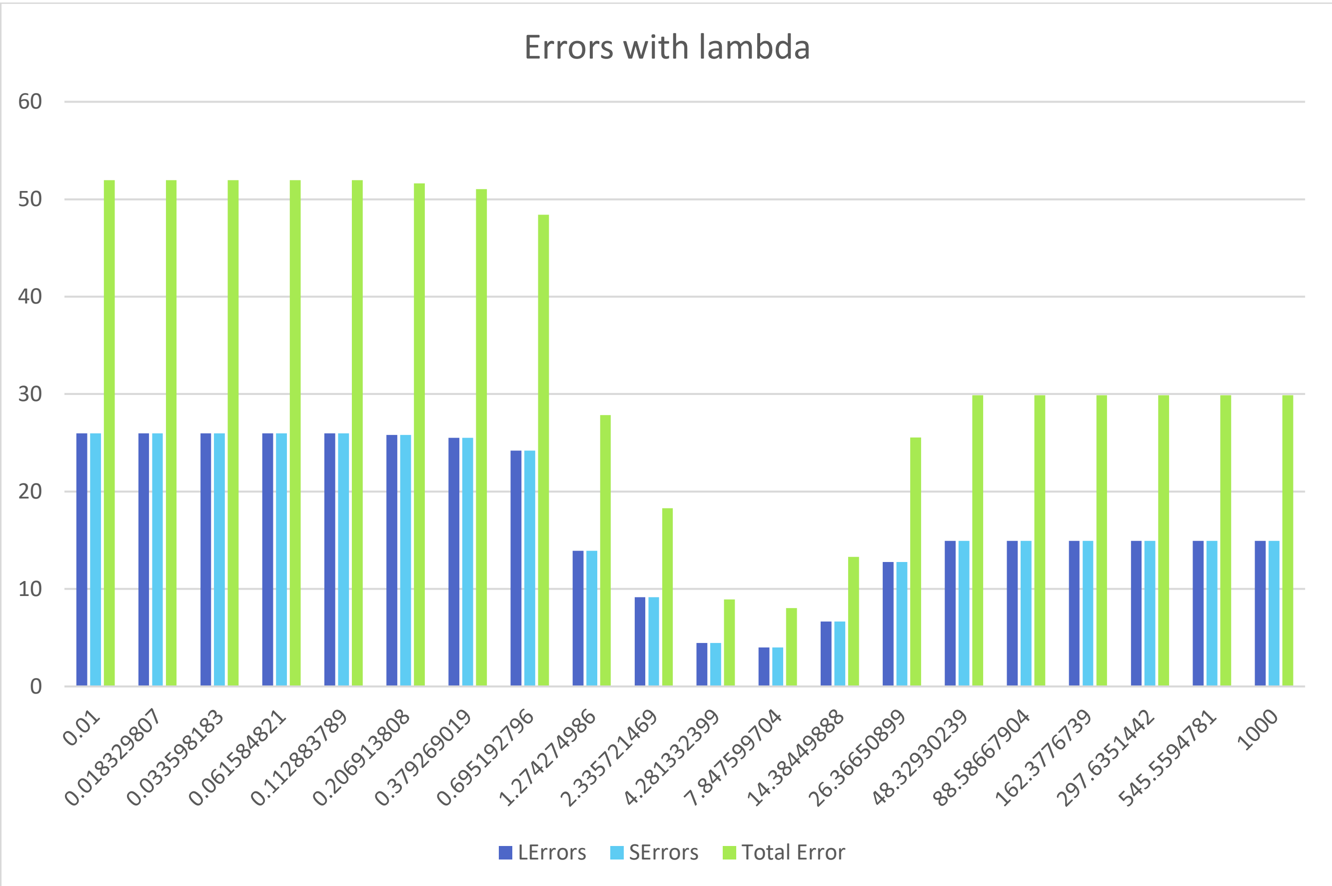
Remarks

As pointed out by others in class, the first part of Question 4 has not been solvable using our computers. It was even not solvable on Google Colab. Thus we have not taken up this subsection. We instead move straight to the second subsection

Results

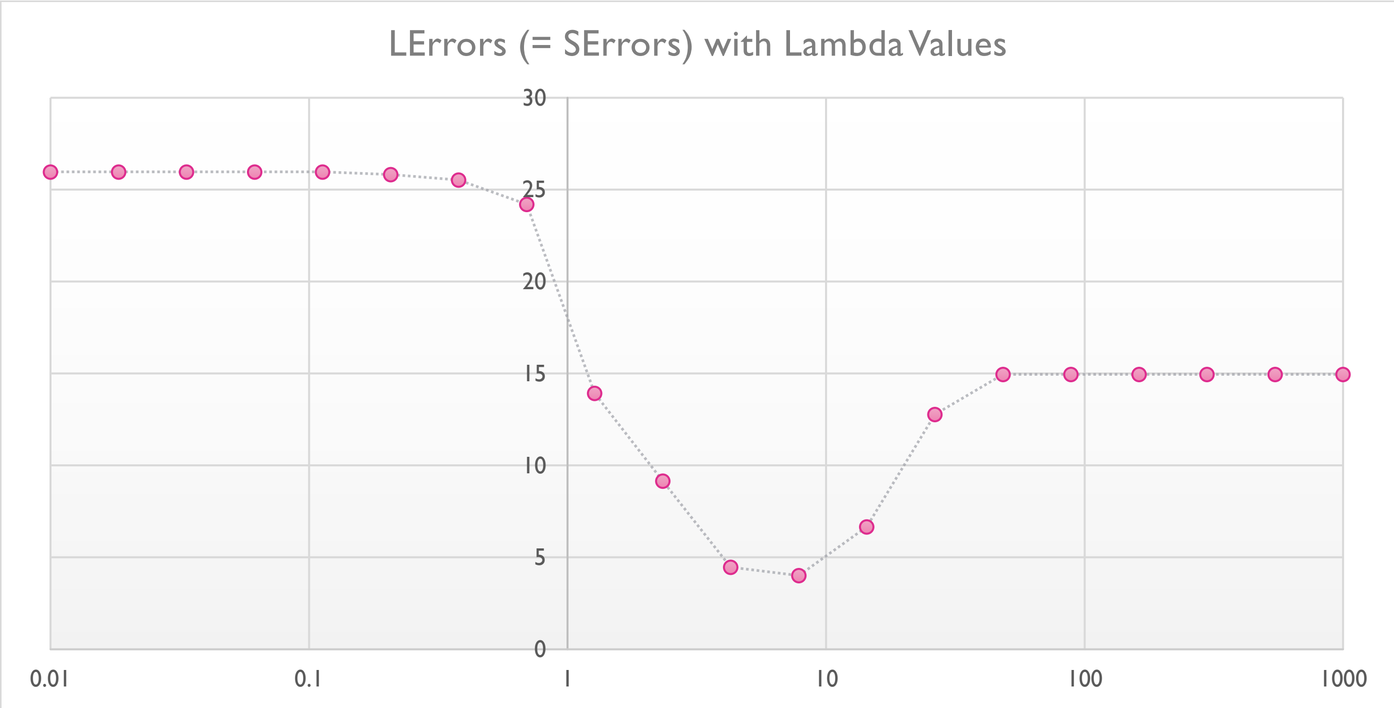
Part 2

We now indicate the errors with various lambda values

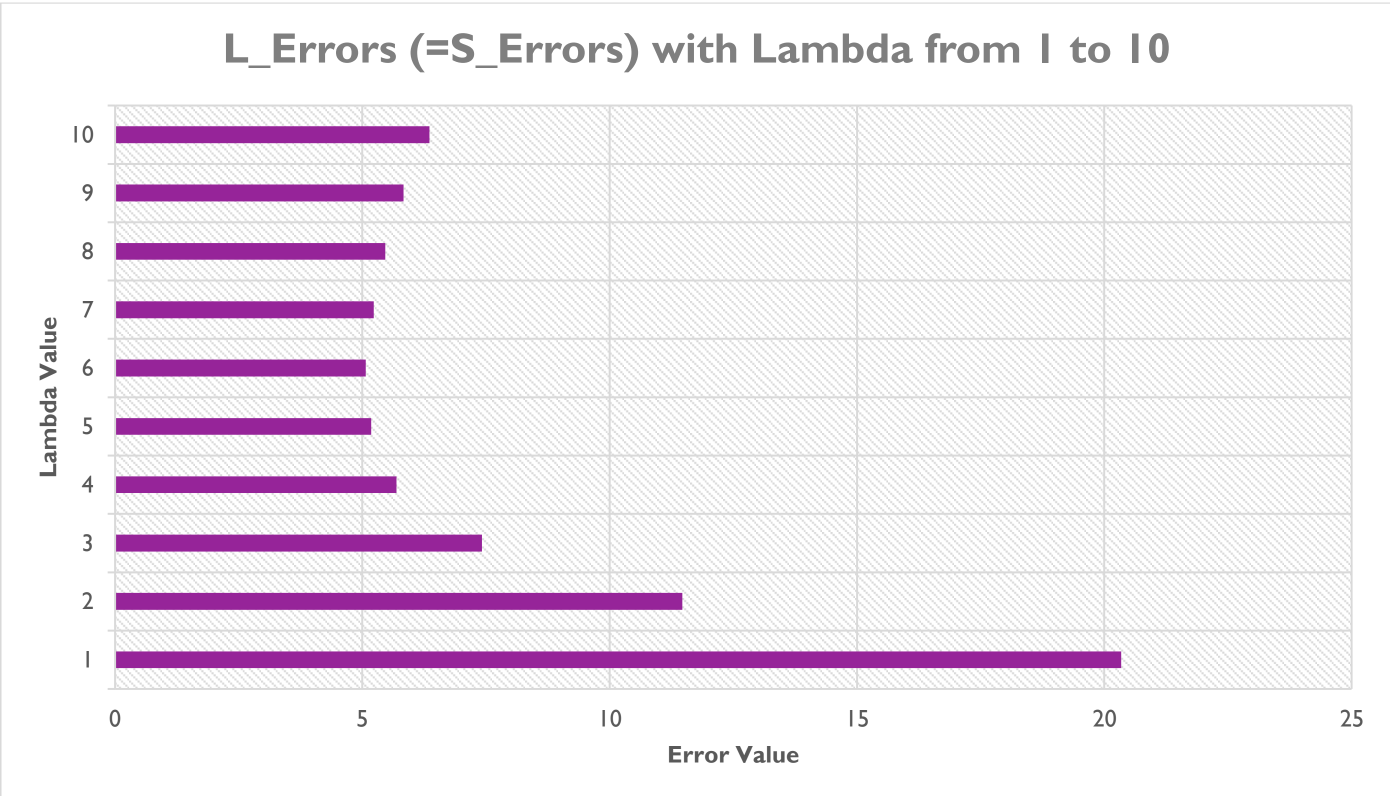


We first note that the errors between the L matrices and S matrices have the same norm but opposing signs. This is because the L and S matrices sum to a constant and thus their difference also would be the same. Thus from now we only look at LError

We plot the LError as below



We deep dive in the part where errors are low, ie from 1 to 10.



Question 5

Challenges

- This section was theoretically the most challenging as we had to model a graph through an LP
- I recreated the rgg function in python so that we could run the code easily
- I wasted many hours on the problem as I was solving min-cut, instead of max-cut, which happens to be an equally interesting problem from the computer science stand-point!

Part 1 – Prove that the max-cut problem can be expressed as an SDP

The weight of a cut wrt a subset $S \subseteq V$ is given by

$$w(S) = \sum_{\substack{i \in S \\ j \in S^c}} w_{ij}$$

Let us assign a label $x_i \in \{-1, 1\}$ to each vertex of the graph $G = (V, E)$ wrt a subset $S \subseteq V$:

$$x_i = \begin{cases} 1 & \text{if } i \in S \\ -1 & \text{if } i \in S^c \end{cases}$$

MAX CUT - Problem

$$\text{Maximize} \quad \frac{1}{4} \sum_{i<j} w_{ij} (x_i - x_j)^2$$

$$\text{Subject To} \quad x_i \in \{1, -1\} \quad \text{for } i = 1 - n$$

The Laplacian for a given undirected weighted simple graph $G = (V, E)$ is defined as

$$L(i, j) = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{k=1}^{n} w_{ik} & \text{if } i = j \end{cases}$$

Note that

1. $L \in S_{+}^{n}$
2. Row sums & column sums are 0

Then we can simply rewrite

$$\sum_{i<j} w_{ij} (x_i - x_j)^2 = x^T L x$$

where L was as given in the previous section.

Further we have $x_i \in \{1, -1\} \ \forall i$.

and ① Rank $(X) = 1$

② $x^T L x = \text{Trace}(L X)$ where $X = x x^T \in S_+^n$

③ $x_i \in \{1, -1\} \Rightarrow X(i,i) = 1 \ \forall i$

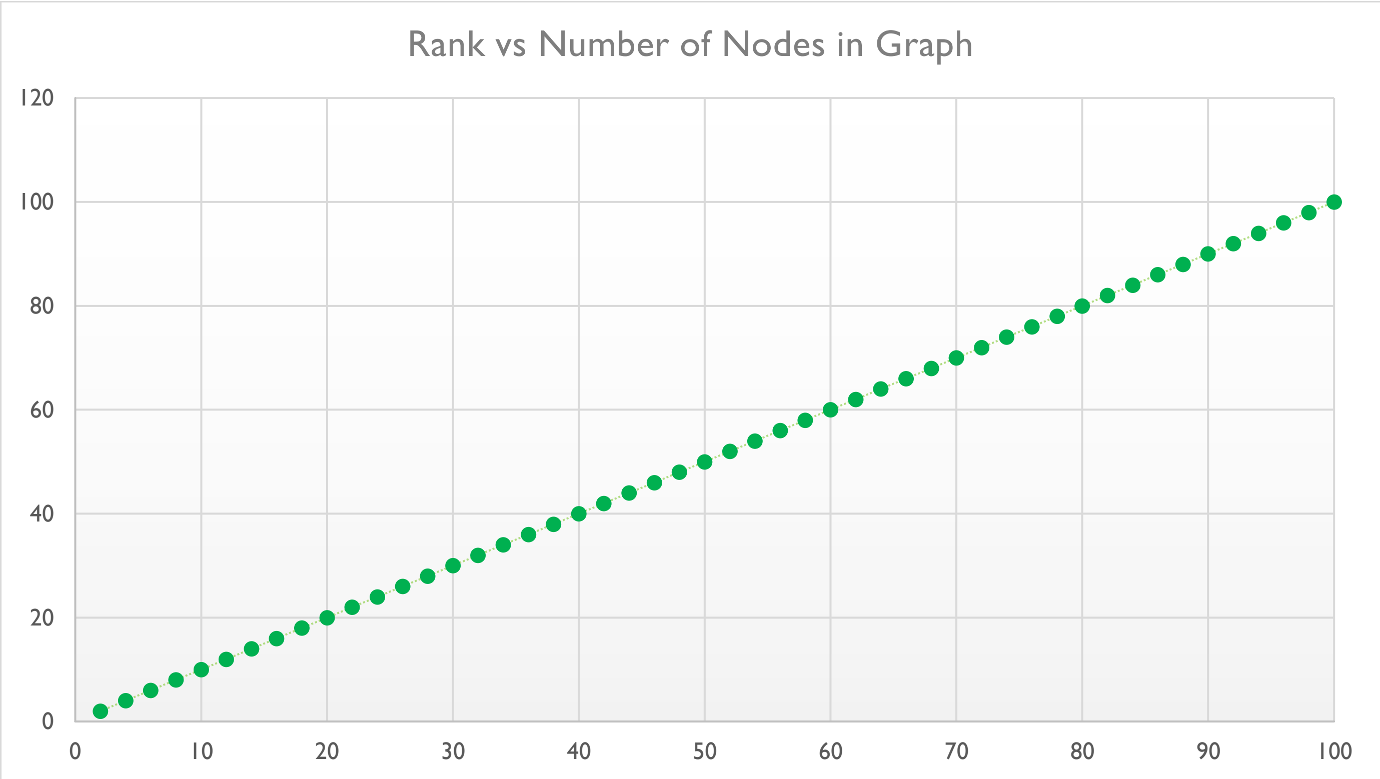
Thus our equivalent formulation is

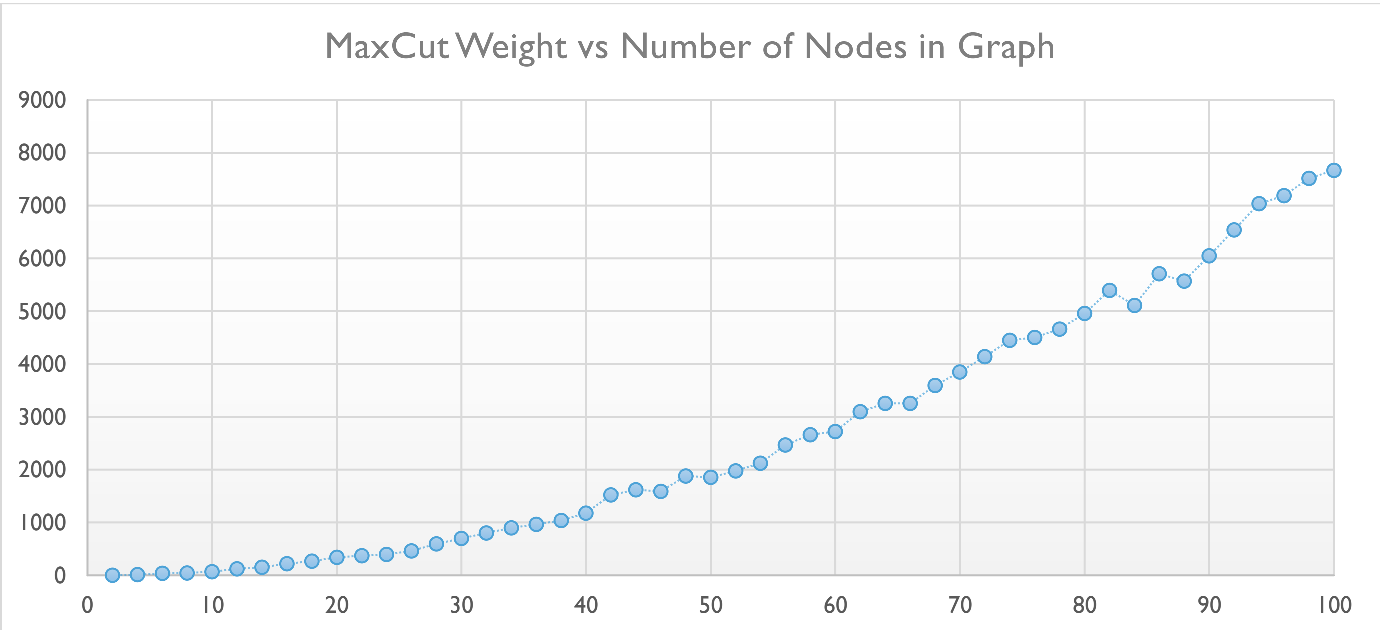
$$\boxed{\begin{array}{ll} \text{Maximize} & \frac{1}{4} \langle L, X \rangle \\ \text{Subject to} & X \in S_+^n \\ & \text{Rank}(X) = 1 \\ & X(i,i) = 1 \ \forall i \end{array}}$$

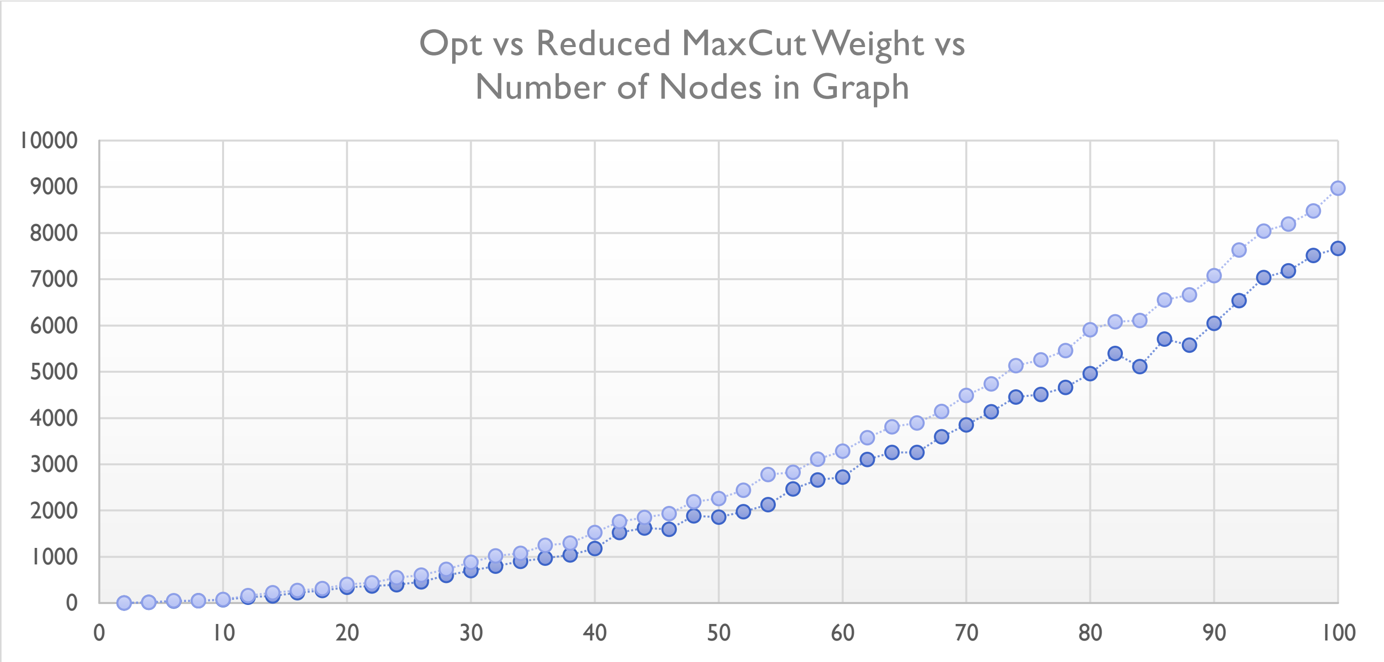
Answer – part 2

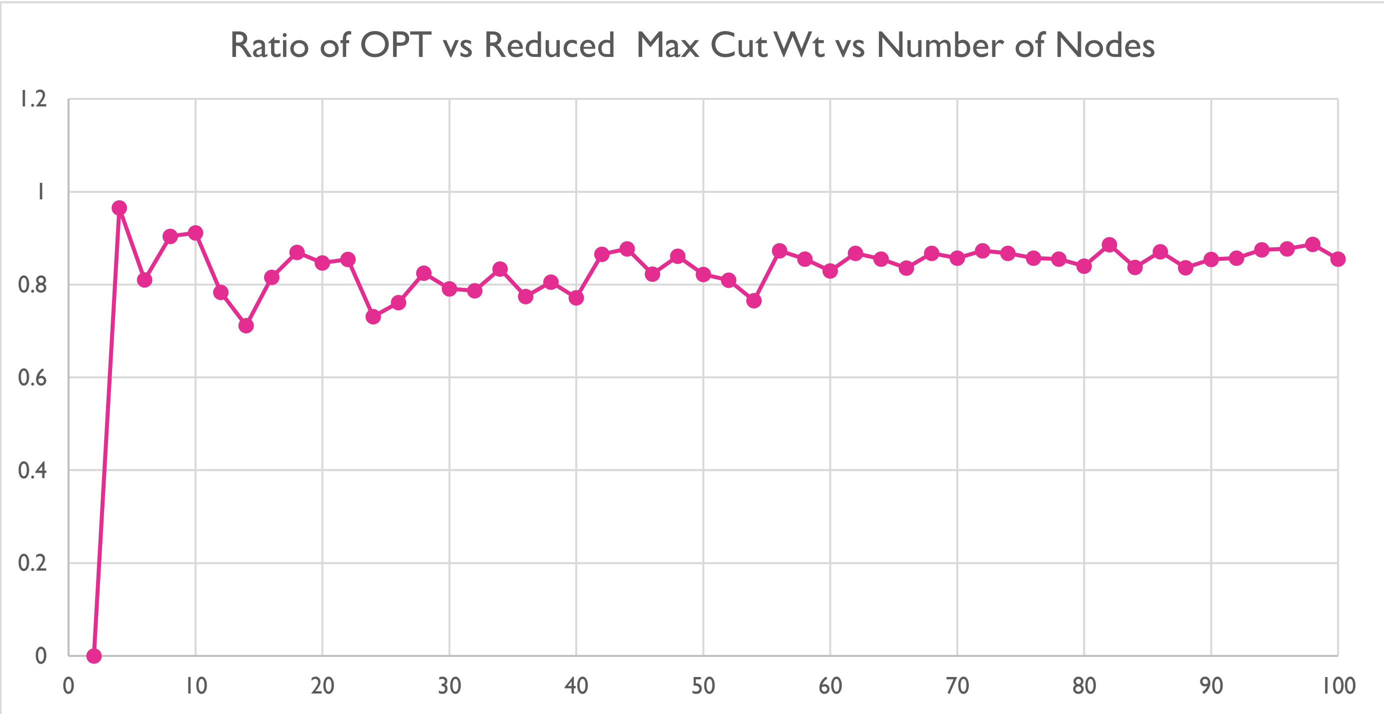
- In part 1 we are asked to comment on the tightness of the approximation if the rank is 1. We can say that if the rank is 1, we have an exact approximation.
- In other words, we say that the rounding will give an exact match with the SDP OPTimum solution
- This is because, since there is only rank 1 in the SDP solution, the rounding with the random vector is able to replicate an exact cut on the real adjacency matrix
- Thus when we see a rank 1 approximation, we can say that the rounding gives exact solution

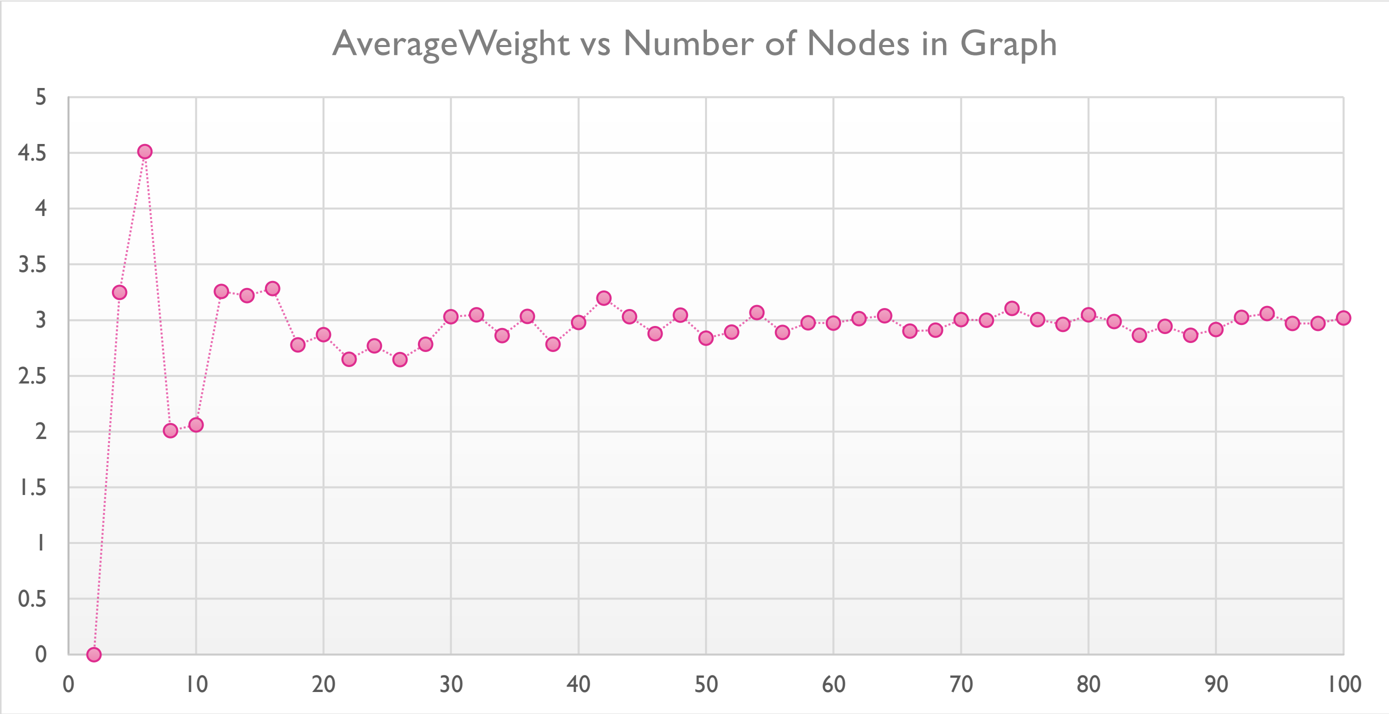
Results Part 2






Opt vs Reduced MaxCut Weight vs
Number of Nodes in Graph
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
60
70
80
90
100


Ratio of OPT vs Reduced Max Cut Wt vs Number of Nodes
1.2
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
80
90
100

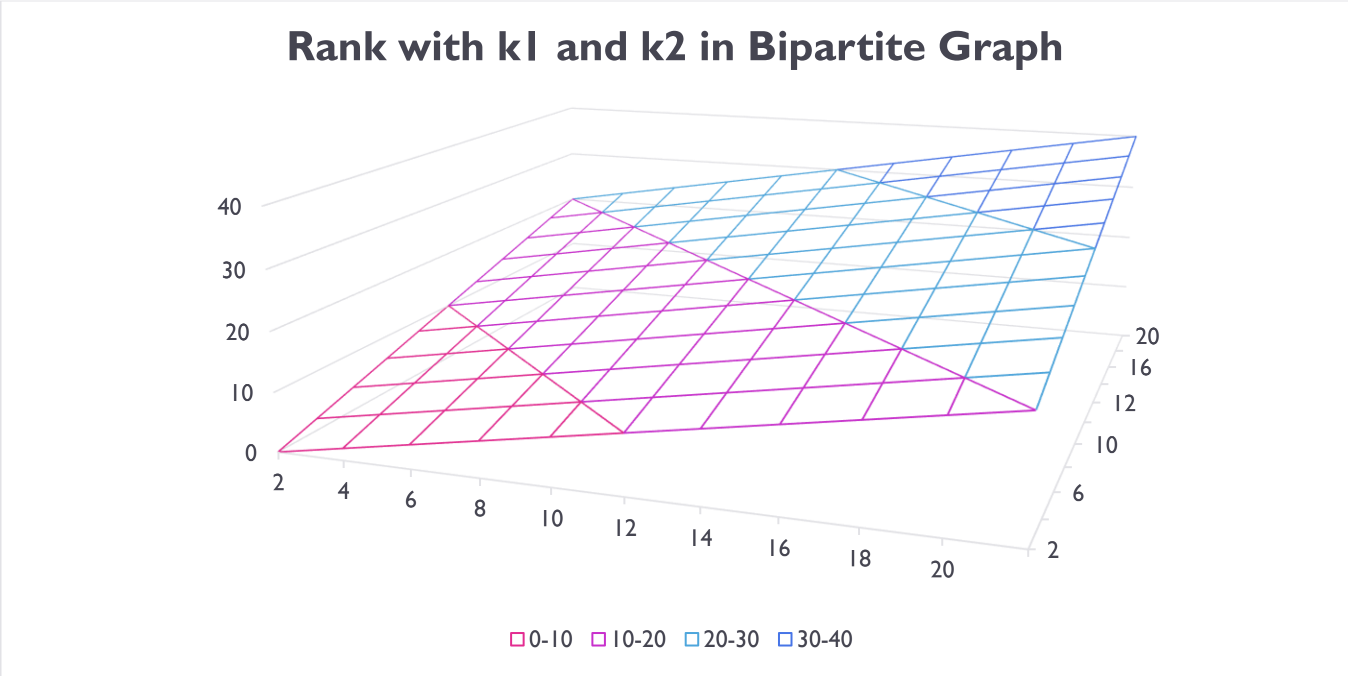


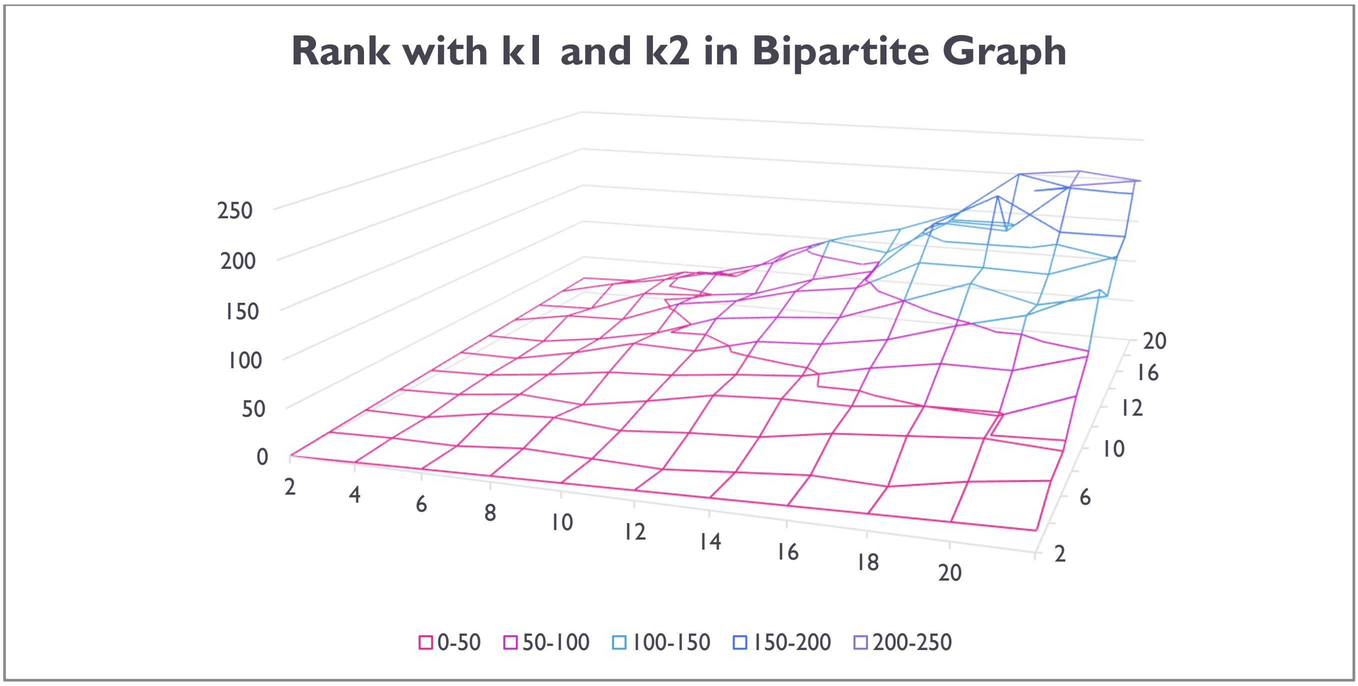
Answers Part 3

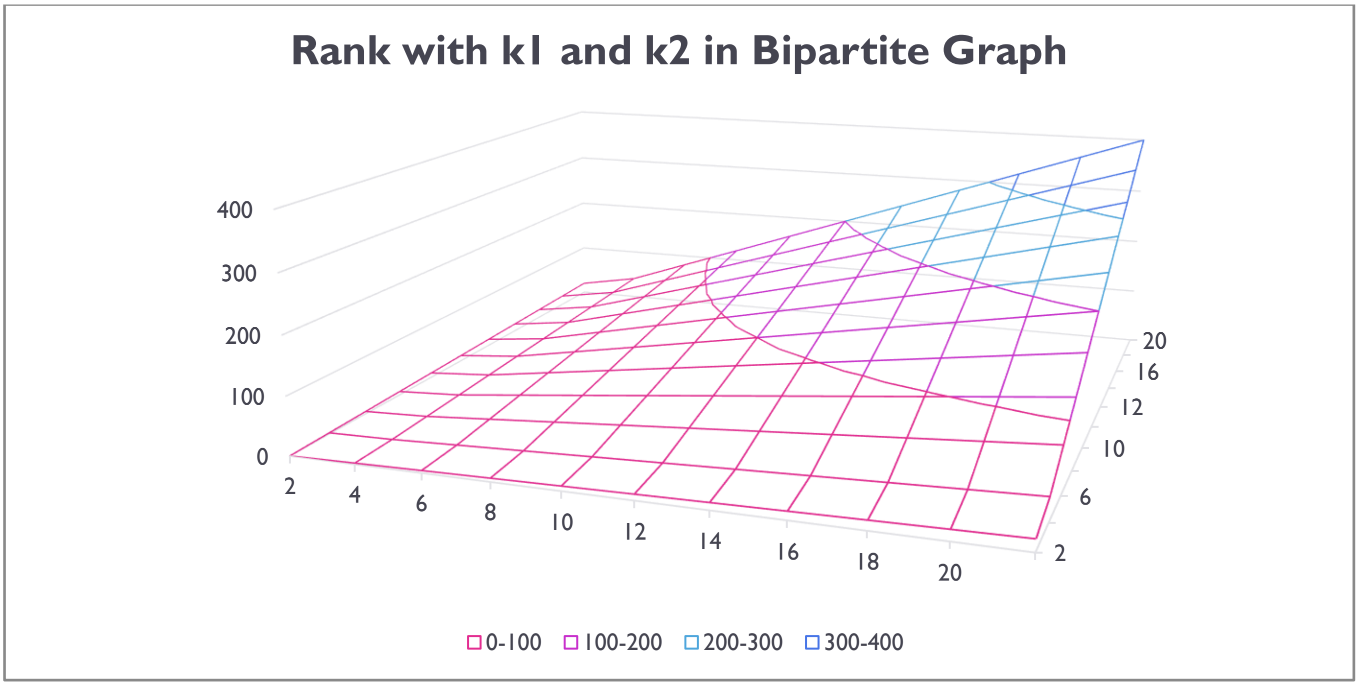
- The optimal cut In a complete bipartite K(m,n) graph is clearly m * n
- This is because we can cut through the center of the graph
- Our solutions indicate that we never get a rank 1 matrix
- Thus all we have is an approximation to the solution. We plot the approximation ratios below and see that it lies at theoretical bounds in expectation
- We therefore suggest that the result is tight in expectation with a approximation ratio of 0.87856 which is as per theory

Results Part 3

We now show a surface plot of what happens in a bipartite graph. We can see that in a $K_{n,m}$ bipartite, the max-cut will have to be m*n as we can cut through the center of the bipartite.






Rank with k1 and k2 in Bipartite Graph
400
300
200
100
0
2
4
6
8
10
12
14
16
18
20
2
6
10
12
16
20
0-100
100-200
200-300
300-400

Problem 6

Remarks

- This was one of the easiest problems in the question paper. And didn't take long to solve
- The nature of the problem is convex because -xlog(x) is a concave function, which we are maximizing
- Conversely, xlog(x) is a concave function, which we are minimizing

Answers

- **Verify that the optimal distribution is uniform:**
    - For this we can note the following
    - - X log(x) is a concave function
    - X log(x) is a convex function
    - We are trying to minimize this
    - Thus, we can apply Jensen's inequality
    - $(f(p_1) + f(p_2))/2 > f((p_1 + p_2)/2)$
    - More pertinently,
    - $(f(p_1) + f(p_2)+\ldots f(p_n))/n > f((p_1+ p_2+\ldots p_n)/n)$
    - Thus $(p_1+ p_2+\ldots p_n)/n$ is the minimizer of the entropy function when there are n variables to assign the probability to
    - Since the probabilities are uniform, the variates they describe are in uniform distribution

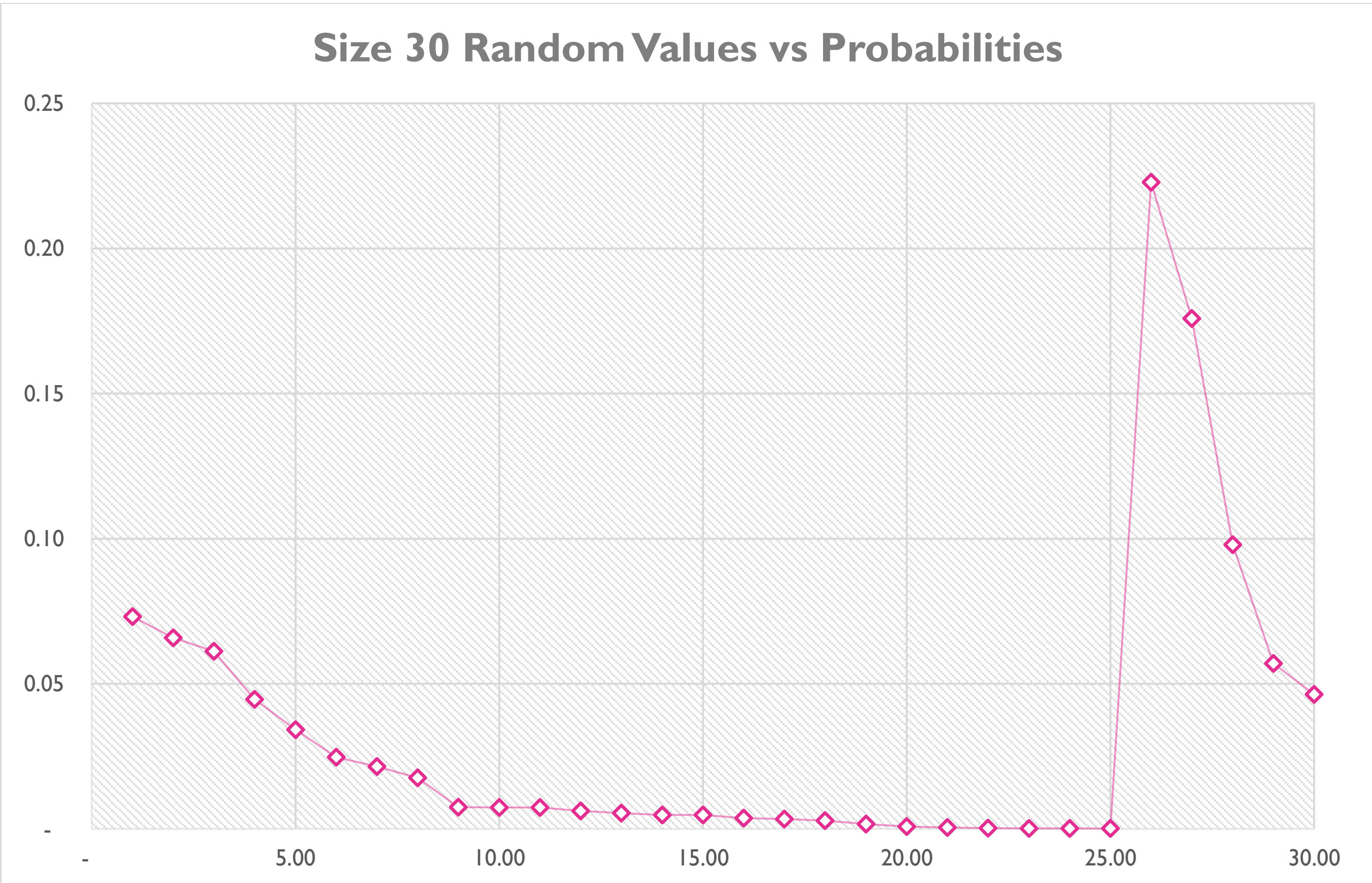
Results

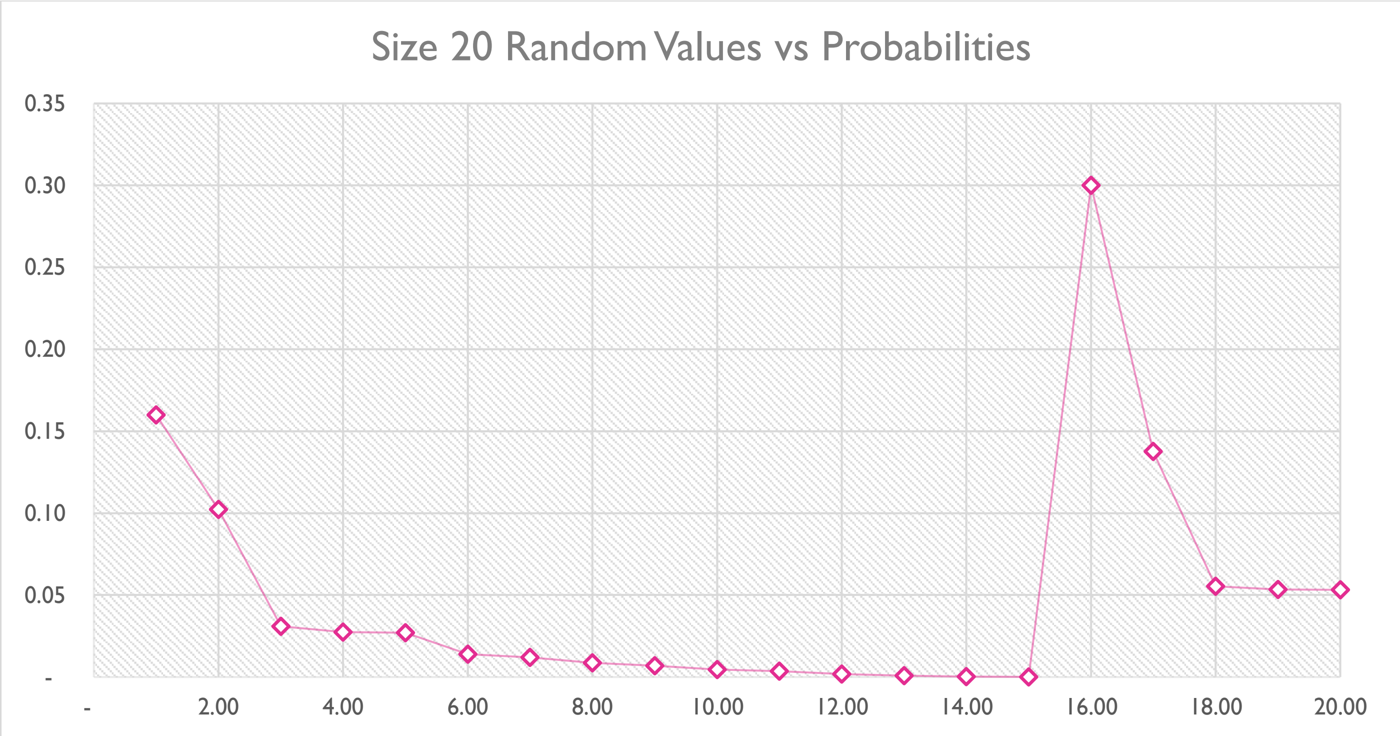
Part 1 Results

We plot the results not just for the variables asked, but for all values between 2 and 20. Notice that the p values are of the form [1/n…1/n]

num Variables	p values->																			
2	0.50	0.50																		
4	0.25	0.25	0.25	0.25																
6	0.17	0.17	0.17	0.17	0.17	0.17														
8	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.13												
10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10										
12	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08								
14	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07						
16	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06				
18	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06		
20	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05

Part 2 Results




Size 20 Random Values vs Probabilities
0.35
0.30
0.25
0.20
0.15
0.10
0.05
-
-
2.00
4.00
6.00
8.00
10.00
12.00
14.00
16.00
18.00
20.00

Packages Required to run

1. NUMPY
2. CVXPY
3. CVXOPT
4. MOSEK
5. SCIPY

Additional Details

Colab Details

Collab was run from this link:

Problem 3

https://colab.research.google.com/drive/1BsZP1Jwx5yu1C10GuLG0KMJ6iClGy3F1?usp=sharing

Problem 3.4

https://colab.research.google.com/drive/1pv4jd0W3GOdX_cqLt318Z8T2s8-6M67u?usp=sharing

Github Details

The entire code is stored here:

https://github.com/p10rahulm/convexOpt

APPENDIX

FILES FOR CODING ASSIGNMENT

1. Question1.py

```
import numpy as np, numpy.random as random, cvxpy as cp
import randomMatrix, utilities
from cvxopt import matrix, solvers


# https://math.stackexchange.com/questions/1639716/how-can-l-1-norm-minimization-with-
linear-equality-constraints-basis-pu
# We formulate the LP as minimize sum(t_i) for |A_ix - b_i| \leq t_i
# or in other words minimize sum(t_i) for (A_ix - b_i) \leq t_i and (A_ix - b_i) \geq -t_i
# Which can be written as minimize 1.T.dot(t) for (A_ix - b_i) \leq t_i and (A_ix - b_i)
\geq -t_i
# OR minimize 0.T.dot(x) + 1.T.dot(t)
# such that
# (A_i x - t_i) \leq +b_i
# \forall i in [n]
# and
# -(A_ix + t_i) \leq -b_i
# and
# -t_i < 0
# \forall i in [n], where x \in R^r, t in R^n
#
# OR
# minimize 0.T.dot(x) + 1.T.dot(t)
# such that
# (A x - I t) \leq b
# and
# -Ax - I t \leq -b
# and
# 0x - I t \leq 0

def minimizeL1NormLP(A, b, n, d):
    opt_coeffs_x = np.zeros((d))
    opt_coeffs_t = np.ones((n))
    opt_coeffs = np.concatenate((opt_coeffs_x, opt_coeffs_t))
    opt_coeffs = matrix(opt_coeffs)
    # The third constraint is actually redundant
    # constraint_coeffs = np.vstack((np.hstack((A, -np.eye(n))),np.hstack((-A, -
np.eye(n))),np.hstack((np.zeros(A.shape), -np.eye(n)))))
    # constraint_offsets = np.concatenate((-b,b,np.zeros(n)))

    constraint_coeffs = np.vstack((np.hstack((A, -np.eye(n))), np.hstack((-A, -
np.eye(n)))))
    constraint_coeffs = matrix(constraint_coeffs)
    constraint_offsets = np.concatenate((b, -b))
    constraint_offsets = matrix(constraint_offsets)
    sol = solvers.lp(opt_coeffs, constraint_coeffs, constraint_offsets)
    out = sol['x'][:d, 0]
    return np.array(out)


# In the above formulation, instead of a vector t, we can just use a scalar t
# Then we write
# minimize t
# such that
# (A_ix - b_i) \leq t
# and
# (A_ix - b_i) \geq -t
# and
# t \geq 0
#
# The problem can be restated as
# minimize t
# such that
```

```python
# (A_ix - t) \leq b_i
# and
# (-A_ix - t) \leq -b_i
# OR
# minimize 0.dot(x) + t
# such that
# (Ax - t1) \leq b
# and
# (-Ax - t1) \leq -b

def minimizeLInfNormLP(A, b, n, d):
    opt_coeffs_x = np.zeros((d))
    opt_coeffs_t = np.ones((1))
    opt_coeffs = np.concatenate((opt_coeffs_x, opt_coeffs_t))
    opt_coeffs = matrix(opt_coeffs)
    # The third constraint is actually redundant
    # constraint_coeffs = np.vstack((np.hstack((A, -np.eye(n))),np.hstack((-A, -
np.eye(n))),np.hstack((np.zeros(A.shape), -np.eye(n)))))
    # constraint_offsets = np.concatenate((-b,b,np.zeros(n)))

    constraint_coeffs = np.vstack((np.hstack((A, -np.ones((n, 1)))), np.hstack((-A, -
np.ones((n, 1))))))
    constraint_coeffs = matrix(constraint_coeffs)
    constraint_offsets = np.concatenate((b, -b))
    constraint_offsets = matrix(constraint_offsets)
    sol = solvers.lp(opt_coeffs, constraint_coeffs, constraint_offsets)
    out = sol['x'][:d, 0]
    return np.array(out)


def minimizeL1NormCVX(A, b, n, d):
    x_out = cp.Variable(d)
    # cvxL1Prob = cp.Problem(cp.Minimize(cp.norm(cp.matmul(A,x_out)-b,1)))
    cvxL1Prob = cp.Problem(cp.Minimize(cp.norm(A @ x_out - b, 1)))
    cvxL1Prob.solve("CVXOPT")
    # x = cp.Variable(d)
    # prob1 = cp.Problem(cp.Minimize(cp.norm(A.dot(x)-b,1)))
    return x_out.value


def minimizeLInfNormCVX(A, b, n, d):
    x_out = cp.Variable(d)
    # cvxL1Prob = cp.Problem(cp.Minimize(cp.norm(cp.matmul(A,x_out)-b,1)))
    cvxL1Prob = cp.Problem(cp.Minimize(cp.norm(A @ x_out - b, np.inf)))
    cvxL1Prob.solve("CVXOPT")
    # x = cp.Variable(d)
    # prob1 = cp.Problem(cp.Minimize(cp.norm(A.dot(x)-b,1)))
    return x_out.value


def fullprint(*args, **kwargs):
    from pprint import pprint
    import numpy
    opt = numpy.get_printoptions()
    numpy.set_printoptions(threshold=numpy.inf)
    pprint(*args, **kwargs)
    numpy.set_printoptions(**opt)


def printResults(A, b, x, typeMessage):
    d = A.shape[1]
    print("\n-----------------------------------------------------------------------------------------
-------------------------\n")
    print(typeMessage)
    print("\n-----------------------------------------------------------------------------------------
```

```
-------------------------\n")
    print("x=", np.ndarray.flatten(x))
    print("\n---------------------------\n")
    Adotxb1 = np.matmul(A, x.reshape(d, 1)) - b.reshape((n, 1))

    adotxprint = np.ndarray.flatten(Adotxb1)
    printOpt = np.get_printoptions()
    np.set_printoptions(threshold=d + 1)
    print("A.dot(x) - b =", adotxprint)
    np.set_printoptions(**printOpt)

    print("L1Norm(A.dot(x)-b) = ", utilities.L1Norm(Adotxb1))
    print("LInfNorm(A.dot(x)-b) = ", utilities.LInfNorm(Adotxb1))

    print("L2Norm(A.dot(x)-b) = ", utilities.L2Norm(Adotxb1))
    print("\n-----------------------------------------------------------------------------
-------------------------\n")


if __name__ == "__main__":
    random.seed(8)
    n = 200
    d = 10
    A = randomMatrix.generateUniformRandomMatrices(n, d)
    b = randomMatrix.generateUniformRandomMatrices(n, 1)[:, 0]
    x = minimizeL1NormLP(A, b, n, d)
    printResults(A, b, x, "Results for l1Norm though LP")
    x = minimizeL1NormCVX(A, b, n, d)
    printResults(A, b, x, "Results for l1Norm though CVXPY")

    x = minimizeLInfNormLP(A, b, n, d)
    printResults(A, b, x, "Results for lInfNorm though LP")

    x = minimizeLInfNormCVX(A, b, n, d)
    printResults(A, b, x, "Results for lInfNorm though CVXPY")
    print("Solutions for the two methods for both L-Inf norma nd L-1 Norm are exactly the
same")
```

2. Question2.py

```
import numpy as np, numpy.random as random, cvxpy as cp, matplotlib.pyplot as plt
import randomMatrix, utilities
import sys


def loss_fn(A, b, x):
    return cp.pnorm(A @ x - b, p=2) ** 2


def regularizer(x, norm=2, pow=2):
    return cp.pnorm(x, p=norm) ** pow


def objective_fn(A, b, x, lmbda, norm=2, pow=2):
    return loss_fn(A, b, x) + lmbda * regularizer(x, norm, pow)


def mse(A, b, x):
    return (1.0 / A.shape[0]) * loss_fn(A, b, x).value


def plot_regularization_path(lmbda_values, x_values):
    num_coeffs = len(x_values[0])
    for i in range(num_coeffs):
        plt.plot(lmbda_values, [wi[i] for wi in x_values])
    plt.xlabel(r"$\lambda$", fontsize=16)
    plt.xscale("log")
    plt.title("Regularization Path")
    plt.show()


def plot_train_test_errors(train_errors, test_errors, lmbda_values):
    plt.plot(lmbda_values, train_errors, label="Train error")
    # plt.plot(lmbda_values, test_errors, label="Test error")
    plt.xscale("log")
    plt.legend(loc="upper left")
    plt.xlabel(r"$\lambda$", fontsize=16)
    plt.title("Mean Squared Error (MSE)")
    plt.show()


def getIterationErrors(problem, tempFile, column):
    originalOut = sys.stdout
    sys.stdout = open(tempFile, "w")
    a = problem.solve(verbose=True)
    sys.stdout.close()
    sys.stdout = originalOut
    f = open(tempFile, "r")
    linenum = 0
    iterationErrors = []
    for line in f:
        linenum += 1
        if linenum < 5:
            continue
        if line == "\n":
            break
        cols = line.split("  ")

        iterationErrors.append(float(cols[column]))
    return iterationErrors


def solveRegression(A, b, m, n, method=0, lmbda=0.1, showIterationErrors=0,
```

```
tempFile="outputs/temp.txt",
                    outFile="outputs/iterations.txt"):
    x = cp.Variable(n)

    if method == 0:
        problem = cp.Problem(cp.Minimize(objective_fn(A, b, x, 0)))
    elif method == 1:
        problem = cp.Problem(cp.Minimize(objective_fn(A, b, x, lmbda, norm=2, pow=2)))
    elif method == 2:
        problem = cp.Problem(cp.Minimize(objective_fn(A, b, x, lmbda, norm=1, pow=1)))
    else:
        problem = cp.Problem(cp.Minimize(objective_fn(A, b, x, 0)))

    if showIterationErrors:
        iterationErrors = getIterationErrors(problem, tempFile, 1)
        f = open(outFile, "w")
        f.write("IterationNumber,Errors\n")
        iterCounter = 1
        for iterError in iterationErrors:
            f.write(str(iterCounter) + "," + str(iterError) + '\n')
            iterCounter += 1
        f.close()

    else:
        problem.solve()
    return x.value


def Part1(A, b, m, n):
    solveRegression(A, b, m, n, method=0, lmbda=0.1, showIterationErrors=1,
tempFile="outputs/temp.txt",
                    outFile="outputs/2.1iterationErrorMethod0.txt")
    solveRegression(A, b, m, n, method=1, lmbda=0.1, showIterationErrors=1,
tempFile="outputs/temp.txt",
                    outFile="outputs/2.1iterationErrorMethod1.txt")
    solveRegression(A, b, m, n, method=2, lmbda=0.1, showIterationErrors=1,
tempFile="outputs/temp.txt",
                    outFile="outputs/2.1iterationErrorMethod2.txt")


def LambdasCoordinatewiseWriteFile(xArray, lambdas, filename, multicoordinates=1):
    f = open(filename, "w")
    if multicoordinates:
        f.write("lambda," + ",".join(("Coordinate" + str(i + 1)) for i in
range(len(xArray[0]))) + "\n")
    else:
        f.write("lambda,value" + "\n")

    for iter in range(len(xArray)):
        x = xArray[iter]
        lmbda = lambdas[iter]
        if multicoordinates:
            f.write(str(lmbda) + "," + ",".join(str(i) for i in x) + "\n")
        else:
            f.write(str(lmbda) + "," + str(x) + "\n")

    f.close()


def Part2(A, b, m, n, lambda_vals):
    method_xs = []
    lambdas = []

    for lmbda in lambda_vals:
        x = solveRegression(A, b, m, n, method=1, lmbda=lmbda, showIterationErrors=0)
```

```python
        method_xs.append(x)
        lambdas.append(lmbda)
    methodFile = "outputs/2.2.1x.txt"
    LambdasCoordinatewiseWriteFile(method_xs, lambdas, methodFile, 1)

    AdotXMinusB = []
    L2Norms = []
    L2NormPlusRegs = []
    for i in range(len(method_xs)):
        x = method_xs[i]
        lmbda = lambdas[i]
        # normTwoSq = loss_fn(A, b, x)
        normTwoSq = utilities.L2Norm(A @ x - b.reshape(m, 1))
        regularizer = lmbda * (utilities.L2Norm(x))
        normTwoSqPlusReg = normTwoSq + regularizer
        AdotXMinusB.append(normTwoSq)
        L2Norms.append(regularizer)
        L2NormPlusRegs.append(normTwoSqPlusReg)
    methodFile = "outputs/2.2.1L2Adotxminusb.txt"
    LambdasCoordinatewiseWriteFile(AdotXMinusB, lambdas, methodFile, 0)

    methodFile = "outputs/2.2.1L2regularizer.txt"
    LambdasCoordinatewiseWriteFile(L2Norms, lambdas, methodFile, 0)

    methodFile = "outputs/2.2.1L2NormPlusRegularizer.txt"
    LambdasCoordinatewiseWriteFile(L2NormPlusRegs, lambdas, methodFile, 0)

    method_xs = []
    lambdas = []

    for lmbda in lambda_vals:
        x = solveRegression(A, b, m, n, method=2, lmbda=lmbda, showIterationErrors=0)
        method_xs.append(x)
        lambdas.append(lmbda)
    methodFile = "outputs/2.2.2x.txt"
    LambdasCoordinatewiseWriteFile(method_xs, lambdas, methodFile)

    AdotXMinusB = []
    L2Norms = []
    L2NormPlusRegs = []
    for i in range(len(method_xs)):
        x = method_xs[i]
        lmbda = lambdas[i]
        # normTwoSq = loss_fn(A, b, x)
        normTwoSq = utilities.L2Norm(A.dot(x) - b.reshape(m, 1))
        regularizer = lmbda * (utilities.L1Norm(x))
        normTwoSqPlusReg = normTwoSq + regularizer
        AdotXMinusB.append(normTwoSq)
        L2Norms.append(regularizer)
        L2NormPlusRegs.append(normTwoSqPlusReg)
    methodFile = "outputs/2.2.2L1Adotxminusb.txt"
    LambdasCoordinatewiseWriteFile(AdotXMinusB, lambdas, methodFile, 0)

    methodFile = "outputs/2.2.2L1regularizer.txt"
    LambdasCoordinatewiseWriteFile(L2Norms, lambdas, methodFile, 0)

    methodFile = "outputs/2.2.1L1NormPlusRegularizer.txt"
    LambdasCoordinatewiseWriteFile(L2NormPlusRegs, lambdas, methodFile, 0)


def Part3(m, n, lambda2, lambda3):
    sigma_min = 0.1
    sigma_step = 0.1
    method1Errors=[]
    method2Errors = []
```

```python
    method3Errors = []
    sigmas = []
    for i in range(20):
        sigma = sigma_min + i * sigma_step
        sigmas.append(sigma)
        A, b = randomMatrix.gendata_lasso(m, n,sigma,1)
        b = np.ndarray.flatten(b)
        x = solveRegression(A, b, m, n, method=0)
        rmse_error = np.sqrt(mse(A, b, x))
        method1Errors.append(rmse_error)

        x = solveRegression(A, b, m, n, method=1, lmbda=lambda2)
        rmse_error = np.sqrt(mse(A, b, x))
        method2Errors.append(rmse_error)

        x = solveRegression(A, b, m, n, method=2, lmbda=lambda3)
        rmse_error = np.sqrt(mse(A, b, x))
        method3Errors.append(rmse_error)
    f = open("outputs/2.iii.txt","w")
    f.write("Sigma,RMSE_LSQ,RMSE_L2Norm,RMSE_L1Norm\n")
    for i in range(20):
        sigma = sigmas[i]
        m1e = method1Errors[i]
        m2e = method2Errors[i]
        m3e = method3Errors[i]
        f.write(str(sigma)+","+str(m1e)+","+str(m2e)+","+str(m3e)+"\n")

    f.close()

def Part4(m, n, lambda2, lambda3):
    sigma_min = 0.1
    sigma_step = 0.1
    method1Errors=[]
    method2Errors = []
    method3Errors = []
    sigmas = []
    for i in range(20):
        sigma = sigma_min + i * sigma_step
        sigmas.append(sigma)
        A, b = randomMatrix.gendata_lasso(m, n,sigma,2)
        b = np.ndarray.flatten(b)
        x = solveRegression(A, b, m, n, method=0)
        rmse_error = np.sqrt(mse(A, b, x))
        method1Errors.append(rmse_error)

        x = solveRegression(A, b, m, n, method=1, lmbda=lambda2)
        rmse_error = np.sqrt(mse(A, b, x))
        method2Errors.append(rmse_error)

        x = solveRegression(A, b, m, n, method=2, lmbda=lambda3)
        rmse_error = np.sqrt(mse(A, b, x))
        method3Errors.append(rmse_error)
    f = open("outputs/2.iv.txt","w")
    f.write("Sigma,RMSE_LSQ,RMSE_L2Norm,RMSE_L1Norm\n")
    for i in range(20):
        sigma = sigmas[i]
        m1e = method1Errors[i]
        m2e = method2Errors[i]
        m3e = method3Errors[i]
        f.write(str(sigma)+","+str(m1e)+","+str(m2e)+","+str(m3e)+"\n")

    f.close()
```

```python
if __name__ == "__main__":
    random.seed(8)
    m = 200
    n = 10
    A, b = randomMatrix.gendata_lasso(m, n)
    b = np.ndarray.flatten(b)
    # print("A.shape=", A.shape)
    # print("b.shape", b.shape)

    Part1(A, b, m, n)
    lambda_vals = np.logspace(-2, 3, 50)
    # lambda_vals = range(5, 105, 5)
    Part2(A, b, m, n, lambda_vals)

    # For Part 3, we will use lambda = 2.25 for Part 2 and lambda = 0.1 for part 3

    Part3(200, 50, 2.25, 0.1)

    Part4(200, 50, 2.25, 0.1)
```

3. Question3.py

```
import randomMatrix, utilities
import numpy as np, numpy.random as random, cvxpy as cp
from math import exp


def svm_gendata(Np, Nn, distance):
    Xp = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ np.random.randn(2, Np)
    Xp[0, :] = Xp[0, :] + distance
    Xp[1, :] = Xp[1, :] - distance
    yp = np.ones(Np)

    Xn = np.array([[2, -1], [2, 1]]) / np.sqrt(2) @ np.random.randn(2, Nn)
    Xn[0, :] = Xn[0, :] - distance
    Xn[1, :] = Xn[1, :] + distance

    yn = - np.ones(Nn)

    X = np.hstack((Xp, Xn))
    y = np.hstack((yp, yn))

    return X, y


def runforNum(numPos, NumNeg, distance=2.5, verboseRes=False):
    numPos = 50
    numNeg = 50
    distance = 3
    X, y = svm_gendata(numPos, numNeg, distance)
    m = numPos + numNeg
    one = np.ones(m)
    lambd = cp.Variable(m)
    constraints = [lambd >= 0, lambd.T @ y == 0]
    Y = np.diag(y)
    sigma = X.T @ X

    obj = cp.Maximize(lambd.T @ one + cp.quad_form(lambd, -Y @ sigma @ Y) / 2)
    prob = cp.Problem(obj, constraints)
    opt = prob.solve(solver="CVXOPT", verbose=verboseRes)
    if verboseRes:
        print("optimal value", opt)
        print("lambda values are ", lambd.value)
        objectiveVal = lambd.value.T @ one - 0.5 * lambd.value.T @ Y @ sigma @ Y @
lambd.value
        print("opt = ", opt, "obj = ", objectiveVal)
    return opt


# 3.2
def Part2(numPos, numNeg, sizeRange):
    runforNum(numPos, numNeg, True)
    opts = []
    f = open("Q3.1.txt", "w")
    f.write("numPoints,separation,optimal\n")
    for i in sizeRange:
        for j in range(1):
            min_distance = 2.5
            distance_step = 0.5
            distance = min_distance + j * distance_step
            opt = runforNum(i, i, distance, False)
            # print(i,",",opt)
            f.write(str(i) + "," + str(distance) + "," + str(opt) + "\n")
            opts.append(opt)
    # print(opts)
```

```
    f.close()


def compute_K(x1, x2, sigma):
    return exp((-np.linalg.norm(x1 - x2) ** 2) / (sigma * sigma))


def compose_K_sigma(X, sigma, m):
    K = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            K[i, j] = compute_K(X.T[i], X.T[j], sigma)
    return K


def getError(X, sigma, lambd, y, m):
    # First count the negs
    maxIterate = -10 ** 3
    for i in range(m):
        if y[i] == -1:
            sum1 = 0
            for j in range(m):
                sum1 += lambd.value[j] * y[j] * compute_K(X.T[i], X.T[j], sigma)
            if sum1 > maxIterate:
                maxIterate = sum1

    # First count the positive pts
    minIterate = 10 ** 3
    for i in range(m):
        if y[i] == 1:
            sum1 = 0
            for k in range(m):
                sum1 += lambd.value[k] * y[k] * compute_K(X.T[i], X.T[k], sigma)
            if sum1 < minIterate:
                minIterate = sum1

    # Set b as mid
    b = -(maxIterate + minIterate) / 2

    # We expect this to be 0 anyways
    errors = 0
    for i in range(m):
        finalsum = 0
        for j in range(m):
            finalsum += lambd.value[j] * y[j] * compute_K(X.T[i], X.T[j], sigma)
        pred = finalsum + b
        if np.sign(pred) != y[i]:
            errors += 1
    return errors


# 3.3
def Part3(numPos, numNeg, distance):
    X, y = svm_gendata(numPos, numNeg, distance)
    m = numPos + numNeg
    # sigmas = np.array([10**-2, 10**-1, 0.5, 10, 10**2])
    sigmas = np.logspace(-2, 3, 100)
    lambd = cp.Variable(m)
    one = np.ones(m)
    constraints = [lambd >= 0, lambd.T @ y == 0]
    train_errors = []
    lambda_values = []
    opts = []
    sigmasArr = []
    for i in range(len(sigmas)):
```

```
        sigma = sigmas[i]
        Sigma = compose_K_sigma(X, sigma, m)
        obj = cp.Maximize(lambd.T @ one + cp.quad_form(lambd, -Y @ Sigma @ Y) / 2)
        prob = cp.Problem(obj, constraints)
        opt = prob.solve()
        print("optimal value", opt)
        # print("lambda values are ", lambd.value)
        sigmasArr.append(sigma)
        opts.append(opt)
        lambda_values.append(lambd.value)
        tError = getError(X, sigma, lambd, y, m)

        train_errors.append(tError / m)
        print(train_errors[i])

    f = open("Q3.2.txt", "w")
    f.write("sigma,error,tError\n")
    for i in range(len(sigmasArr)):
        sigma = sigmasArr[i]
        opt = opts[i]
        tError = train_errors[i]
        f.write(str(sigma) + "," + str(opt) + "," + str(tError) + "\n")

    # print(opts)
    f.close()


if __name__ == "__main__":
    seed = 10
    numPos = 50
    numNeg = 50
    sizeRange = range(2, 101, 2)
    Part2(numPos, numNeg, sizeRange)
    Part3(numPos, numNeg, 2.5)
```

4. Question3.4.py

```
import numpy as np, cvxpy as cp
from scipy.io import loadmat
from math import exp

def LoadData():
    imageTrain = loadmat('QuestionFiles/imageTrain.mat')['imageTrain'].reshape(784, 5000)
    labelTrain =
np.squeeze(np.array(loadmat('QuestionFiles/labelTrain.mat')['labelTrain']))

    imageTest = loadmat('QuestionFiles/imageTest.mat')['imageTest'].reshape(784, 500)
    labelTest = np.squeeze(np.array(loadmat('QuestionFiles/labelTest.mat')['labelTest']))
    return imageTrain, labelTrain, imageTest, labelTest


def getDataForDigit(imageTrain, labelTrain, imageTest, labelTest, digit):
    DigitTrain = []
    for i in range(len(labelTrain)):
        if labelTrain[i] == digit:
            DigitTrain.append(imageTrain[:, i])

    lenTrain = len(DigitTrain)
    DigitTest = []
    for i in range(len(labelTest)):
        if labelTest[i] == digit:
            DigitTest.append(imageTest[:, i])

    lenTest = len(DigitTest)
    return DigitTrain, lenTrain, DigitTest, lenTest


def getTwoDigitsTrainTest(digit1, digit2, imageTrain, labelTrain, imageTest, labelTest):
    DigitTrain6, lenTrain6, DigitTest6, lenTest6 = getDataForDigit(imageTrain, labelTrain,
imageTest, labelTest, digit1)

    DigitTrain8, lenTrain8, DigitTest8, lenTest8 = getDataForDigit(imageTrain, labelTrain,
imageTest, labelTest, digit2)

    numOnenumTwoTrainLabel = np.hstack((-np.ones(lenTrain6), np.ones(lenTrain8)))
    numOnenumTwoTestLabel = np.hstack((-np.ones(lenTest6), np.ones(lenTest8)))
    numOnenumTwo_TrainImage = np.array(DigitTrain6 + DigitTrain8)
    numOnenumTwo_TestImage = np.array(DigitTest6 + DigitTest8)
    return numOnenumTwoTrainLabel, numOnenumTwoTestLabel, numOnenumTwo_TrainImage,
numOnenumTwo_TestImage


def getAccuracyBetweenDigits(digit1, digit2, imageTrain, labelTrain, imageTest, labelTest):

    numOnenumTwoTrainLabel, numOnenumTwoTestLabel, numOnenumTwo_TrainImage,
numOnenumTwo_TestImage = getTwoDigitsTrainTest(
        digit1, digit2, imageTrain, labelTrain, imageTest, labelTest)
    # print(numOnenumTwo_TrainImage.shape)
    # print(numOnenumTwo_TestImage.shape)

    n = numOnenumTwo_TrainImage.shape[1]
    W = cp.Variable((n))
    b = cp.Variable()
    ones = np.array(np.ones(numOnenumTwo_TrainImage.shape[0]))
    Y = np.diag(numOnenumTwoTrainLabel)

    obj = cp.Minimize((cp.pnorm(W, p=2) ** 2) / 2)
    constraints = [ones - Y @ (numOnenumTwo_TrainImage @ W - b * ones) <= 0]
    prob = cp.Problem(obj, constraints)
```

```
    prob.solve()

    W_final = W.value
    b_final = b.value
    # print(W_final.shape)
    errors = 0
    ones_test = ones = np.array(np.ones(numOnenumTwo_TestImage.shape[0]))
    # numOnenumTwoTestImage@W_final +b*ones
    pred = np.sign(numOnenumTwo_TestImage @ W_final + b_final * ones_test)
    # pred
    inAcc = (np.sign(numOnenumTwo_TestImage @ W_final + b_final * ones_test) !=
numOnenumTwoTestLabel).sum() / \
            numOnenumTwo_TestImage.shape[0]
    acc = 1 - inAcc
    return acc

def compute_K(x1, x2, sigma):
    return exp((-np.linalg.norm(x1 - x2) ** 2) / (sigma * sigma))


def compose_K_sigma(X, sigma,m):
    K = np.zeros((m, m))
    for i in range(m):
        for j in range(m):
            K[i, j] = compute_K(X.T[i], X.T[j], sigma)
    return K

def getAccuracyBetweenDigitsGaussianKerner(digit1, digit2, imageTrain, labelTrain,
imageTest, labelTest):


    # Variable
    trainNum1Num2_label, testNum1Num2_label, trainNum1Num2_image, testNum1Num2_image =
getTwoDigitsTrainTest(
        digit1, digit2, imageTrain, labelTrain, imageTest, labelTest)


    y = trainNum1Num2_label
    X = trainNum1Num2_image.T
    Y = np.diag(trainNum1Num2_label)
    m = y.shape[0]
    one = np.ones(m)
    lambd = cp.Variable(m)
    y_test = testNum1Num2_label
    X_test = testNum1Num2_image.T
    m_test = y_test.shape[0]


    constraints = [lambd >= 0, lambd.T @ y == 0]
    lambda_values = []
    sigma = 2.5
    Sigma = compose_K_sigma(X,sigma,m)
    obj = cp.Maximize(lambd.T @ one + cp.quad_form(lambd, -Y @ Sigma @ Y) / 2)
    prob = cp.Problem(obj, constraints)
    prob.solve()
    lambda_values.append(lambd.value)
    maxi = -999
    for j in range(m):
        if y[j] == -1:

            sum1 = 0
            for k in range(m):
                sum1 = sum1 + lambd.value[k] * y[k] *
compute_K(np.squeeze(np.array(X.T[j])),
```

```
np.squeeze(np.array(X.T[k])), sigma)
            if sum1 > maxi:
                maxi = sum1

    mini = 1000
    for j in range(m):
        if y[j] == 1:

            sum1 = 0
            for k in range(m):
                sum1 = sum1 + lambd.value[k] * y[k] *
compute_K(np.squeeze(np.array(X.T[j])),

np.squeeze(np.array(X.T[k])), sigma)
            if sum1 < mini:
                mini = sum1

    b = -(maxi + mini) / 2

    errors = 0
    for j in range(m_test):
        finalsum = 0
        for k in range(m):
            finalsum += lambd.value[k] * y[k] *
compute_K(np.squeeze(np.array(X_test.T[j])),
                                                          np.squeeze(np.array(X.T[k])),
sigma)
        pred = finalsum + b
        if np.sign(pred) != y_test[j]:
            errors += 1
    accuracy = 1 - (errors / m_test)
    return accuracy


if __name__ == "__main__":
    imageTrain, labelTrain, imageTest, labelTest = LoadData()
    print(labelTrain)
    # Making required Training Data
    accuracy = getAccuracyBetweenDigits(6, 8, imageTrain, labelTrain, imageTest, labelTest)
    print("accuracy =", accuracy)

    accuracies = []
    num1s = []
    num2s = []
    for i in range(1, 10, 1):
        for j in range(i + 1, 10, 1):
            accuracy = getAccuracyBetweenDigits(i, j, imageTrain, labelTrain, imageTest,
labelTest)
            accuracies.append(accuracy)
            num1s.append(i)
            num2s.append(j)
    filename = "outputs/output3.4.txt"
    f = open(filename, "w")
    f.write("num1,num2,accuracy\n")
    for i in range(len(num1s)):
        num1 = num1s[i]
        num2 = num2s[i]
        accuracy = accuracies[i]
        f.write(str(num1) + "," + str(num2) + "," + str(accuracy) + "\n")
    f.close()

    accuracies = []
    num1s = []
    num2s = []
    for i in range(1, 10, 1):
```

```
        for j in range(i + 1, 10, 1):
            accuracy = getAccuracyBetweenDigitsGaussianKerner(i, j, imageTrain, labelTrain,
imageTest, labelTest)
            print("num1=",i,"num1=",j,"accuracy=",accuracy)
            accuracies.append(accuracy)
            num1s.append(i)
            num2s.append(j)
    filename = "outputs/output3.4Gaussian.txt"
    f = open(filename, "w")
    f.write("num1,num2,accuracy\n")
    for i in range(len(num1s)):
        num1 = num1s[i]
        num2 = num2s[i]
        accuracy = accuracies[i]
        f.write(str(num1) + "," + str(num2) + "," + str(accuracy) + "\n")
    f.close()
```

5. Question4.py

```
import cvxpy as cp, numpy as np, numpy.random as random
from scipy import sparse
import matplotlib.pyplot as plt
import randomMatrix


def runSolver(mConstraint, L_Initial, S_Initial, m, n, lmbda=0.1):
    L = cp.Variable((m, n))
    S = cp.Variable((m, n))
    cost = cp.norm(L, "nuc") + lmbda * cp.norm(S, 1)
    constr = [L + S == mConstraint]
    prob = cp.Problem(cp.Minimize(cost), constr)
    prob.solve("MOSEK")
    lError = np.linalg.norm(L.value - L_Initial, 'fro')
    sError = np.linalg.norm(S.value - S_Initial, 'fro')
    return lError, sError


def generateLowRankMatrixPlusSparseMatrix(m, n, rank, density):
    lInitial = randomMatrix.generateLowRank(m, n, rank)
    sInitial = randomMatrix.sparseRandomNormalMatrix(m, n, density=density)
    M = lInitial + sInitial
    return M, lInitial, sInitial


def getErrors(mConstraint, L_Initial, S_Initial, m, n, lambda_vals):
    LErrors = []
    SErrors = []
    for lmbda in lambda_vals:
        print("Now running for lambda = ", lmbda)
        lError, sError = runSolver(mConstraint, L_Initial, S_Initial, m, n, lmbda)
        LErrors.append(lError)
        SErrors.append(sError)
    return LErrors, SErrors


if __name__ == "__main__":
    m = 50
    n = 50
    lambda_vals = np.logspace(-2, 3, 20)
    lambda_vals = range(1, 11)

    rank = 5
    density = 0.1
    mConstraint, L_Initial, S_Initial = generateLowRankMatrixPlusSparseMatrix(m, n, rank,
density)
    LErrors, SErrors = getErrors(mConstraint, L_Initial, S_Initial, m, n, lambda_vals)
    filename = "outputs/4.2Errors.txt"
    f = open(filename, "w")
    f.write("Lambda,LErrors,SErrors\n")
    for i in range(len(LErrors)):
        lmbda = lambda_vals[i]
        lError = LErrors[i]
        sError = SErrors[i]
        f.write(str(lmbda) + "," + str(lError) + "," + str(sError) + "," +
str(lError+sError) + "\n")
```

6. Question5.py

```
import cvxpy as cp, numpy as np, numpy.random as random
import randomMatrix,utilities


def getLaplacian(adjacencyMatrix):
    degree = np.sum(adjacencyMatrix, axis=0)
    degree = np.diag(degree)
    Laplacian = degree - adjacencyMatrix
    return Laplacian

def solveProblem(Laplacian,numNodes,solver):
    eye = np.ones(numNodes)
    X = cp.Variable((numNodes, numNodes), PSD=True)
    cost = (0.25) * cp.trace(Laplacian @ X)
    constr = [X >> 0, cp.diag(X) == eye]
    prob = cp.Problem(cp.Maximize(cost), constr)
    opt = prob.solve(solver=solver)
    # print("prob=",prob,"opt=",opt,"prob.value=",prob.value)
    return X,prob.value


def getCutWeight(X,adjacencyMatrix,numNodes,method=0):
    # print("X=", X)
    M = np.linalg.cholesky(X)
    # print("M=", M)
    u = np.random.uniform(-1, 1, numNodes)
    u = u / utilities.L2Norm(u)
    # print("u=", u)
    labels = M.T @ u
    # Shortcut for setting all positive to 1 and negative to -1
    labels = (((labels >= 0) * 1) - 0.5) * 2
    # print("labels=", labels)
    cutWt = 0
    for i in range(numNodes):
        for j in range(i + 1, numNodes):
            if labels[i] != labels[j]:
                cutWt = cutWt + adjacencyMatrix[i, j]

    # print("MaxCut = ",cutWt)
    return cutWt


def runSolver(numNodes,method=0,k1=8,k2=12):
    if method==1:
        adjacencyMatrix = randomMatrix.generateCompleteBipartite(k1, k2)
        # print("adjacencyMatrix=\n",adjacencyMatrix)
        numNodes=k1+k2
    else:
        adjacencyMatrix = randomMatrix.rgg(numNodes, density=0.5)
    averageWt = np.sum(adjacencyMatrix) / ((numNodes ** 2 - numNodes))
    Laplacian = getLaplacian(adjacencyMatrix)
    X, opt = solveProblem(Laplacian, numNodes, "CVXOPT")
    rank = np.linalg.matrix_rank(X.value)
    # print("X=\n",X.value)

    cutwt = getCutWeight(X.value, adjacencyMatrix, numNodes)
    print("numNodes = ", numNodes, "averageWt = ", averageWt, "rank = ", rank, "cut weight
= ", cutwt, "opt = ", opt)
    return numNodes,averageWt,rank,cutwt,opt


def PartsRunner(nodesRange,filename,method,k1Range,k2Range):
    random.seed(8)
```

```
    if method==1:
        k1s = []
        k2s = []
    else:
        nodes = []
    opts = []
    averageWts = []
    ranks = []
    cutwts = []
    if method==1:
        for k1 in k1Range:
            for k2 in k2Range:
                numNodes, averageWt, rank, cutwt, opt = runSolver(k1+k2, method, k1, k2)
                k1s.append(k1)
                k2s.append(k2)
                averageWts.append(averageWt)
                ranks.append(rank)
                cutwts.append(cutwt)
                opts.append(opt)

    else:
        for numNodes in nodesRange:
            numNodes, averageWt, rank, cutwt,opt = runSolver(numNodes,method,k1=0,k2=0)
            nodes.append(numNodes)
            averageWts.append(averageWt)
            ranks.append(rank)
            cutwts.append(cutwt)
            opts.append(opt)

    f = open(filename,"w")
    if method==1:
        f.write("k1,k2,AverageWeight,Rank,CutWeight,Opt\n")
        i=0
        for k1 in k1Range:
            for k2 in k2Range:
                k1 = k1s[i]
                k2 = k2s[i]
                averageWt = averageWts[i]
                rank = ranks[i]
                cutwt = cutwts[i]
                opt = opts[i]
                f.write(str(k1) + "," + str(k2) + "," + str(averageWt) + "," + str(rank) +
"," + str(cutwt) + "," + str(
                    opt) + "\n")
                i+=1
    else:
        f.write("numNodes,AverageWeight,Rank,CutWeight,Opt\n")
        i = 0
        for i in range(len(nodes)):
                numNodes = nodes[i]
                averageWt = averageWts[i]
                rank = ranks[i]
                cutwt = cutwts[i]
                opt = opts[i]
                f.write(str(numNodes) + "," + str(averageWt) + "," + str(rank) + "," +
str(cutwt) + "," + str(opt) + "\n")
                i += 1
    f.close()




if __name__ == "__main__":
    # We follow the sequence provided by Prof Chirayu in his notes
```

```
    nodesRange = range(2,101,2)
    filename = "outputs/Q5.2.txt"
    PartsRunner(nodesRange,filename,0,0,0)

    k1Range = range(2,21,2)
    k2Range = range(2,21,2)
    filename = "outputs/Q5.3.txt"

    PartsRunner(nodesRange, filename, 1, k1Range, k2Range)
```

7. Question6.py

```
import numpy as np, cvxpy as cp, numpy.random as random


def Solver(numVars, method=0):
    mOnes = np.ones(numVars)
    a = np.sort(random.uniform(-1, 1, numVars))

    p = cp.Variable(numVars)
    entropy = cp.sum(cp.entr(p))
    if method == 1:
        aSq = np.power(a, 2)
        aExp = 3 * np.power(a, 3) - 2 * a
        aLessPoint5 = [a < 0.5] * 1
        constraints = [p >= 0,
                       cp.matmul(mOnes, p) == 1,
                       p @ a <= 0.1,
                       p @ a >= -0.1,
                       p @ aSq >= 0.5,
                       p @ aSq <= 0.6,
                       p @ aExp >= -0.3,
                       p @ aExp <= -0.2,
                       p @ aLessPoint5 >= 0.3,
                       p @ aLessPoint5 <= 0.4]
    else:
        constraints = [p >= 0, cp.matmul(mOnes, p) == 1]

    prob = cp.Problem(cp.Maximize(entropy), constraints)
    prob.solve()
    print("numVars = ", numVars, "p.value=", p.value)
    return a, p.value


def PartsRunner(filename, numRange, method=0):
    f = open(filename, "w")
    f.write("numVariables,p values->\n")
    f.close()

    for numVars in numRange:
        f = open(filename, "a")
        a, pVals = Solver(numVars, method)

        if pVals is not None:
            f.write(str(numVars) + "-random-vals:" + ",")
            f.write(",".join(str(i) for i in a) + "\n")
            f.write(str(numVars) + "-probabilities:" + ",")
            f.write(",".join(str(i) for i in pVals) + "\n")
        else:
            f.write("Equations Not Satisfied for matrix size:"+str(numVars) + "\n")
        f.close()


if __name__ == "__main__":
    random.seed(8)
    numRange = range(2, 21, 2)
    filename = "outputs/pVals6.1.txt"
    PartsRunner(filename, numRange, 0)
    numRange = range(10, 31, 2)
    filename = "outputs/pVals6.2.txt"
    PartsRunner(filename, numRange, 1)
```